1969-12

An implementation of LISP 1.5 for the IBM 360/67 computer.

Gentry, Donald Gunn

Monterey, California. Naval Postgraduate School

AN IMPLEMENTATION OF LISP 1.5 FOR THE IBM 360/67 COMPUTER

by

Donald Gunn Gentry

United States
Naval Postgraduate School

THESIS

AN IMPLEMENTATION OF LISP 1.5
FOR
THE IBM 360/67 COMPUTER

by

Donald Gunn Gentry

December 1969

An Implementation of LISP 1.5 for The IBM 360/67 Computer

by

Donald Gunn Gentry
Lieutenant, United States Navy
B.I.E., Georgia Institute of Technology, 1962

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1969

ABSTRACT

The design and implementation of the NPS LISP programming system is described. NPS LISP is an interactive version of LISP 1.5, a sophisticated list processing and symbol manipulation computer language. NPS LISP was implemented in PL/I for operation under the CP/CMS timesharing system on the IBM 360/67 computer. It is an interpretive system patterned after 7090 LISP. Most of the features of 7090 LISP are included in NPS LISP.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

I. INTRODUCTION

A member of the family of list processing computer languages is the highly sophisticated language LISP. The LISP language was first developed and implemented at Massachusetts Institute of Technology in connection with research in Artificial Intelligence. It has since been used for symbolic calculations in Differential and Integral Calculus, proving theorems in the Predicate Calculus, game playing, and other areas of Artificial Intelligence.

This thesis describes the development and implementation of an on-line interactive version of LISP for the IBM 360/67 computer. In order to produce a basic LISP system in a reasonably short length of time, it was decided to implement an interpreter-oriented LISP system written in PL/I. The system would operate under the existing CP/CMS time-sharing system. The steps taken to implement NPS LISP[1] were: (1) gather and study available literature concerning LISP systems; (2) decide on the internal system structure; (3) write the elementary LISP functions; (4) write the input and output functions and the interpreter; and (5) write additional procedures which give the system the desired capability in terms of predefined LISP functions.

A basic knowledge of LISP and an understanding of the material in the LISP 1.5 Reference Manual [Ref. 2] is essential to a clear understanding of this thesis. The list representation conventions, abbreviations, and descriptive terms contained herein are those found in Ref. 2. The reader is assumed to have an elementary knowledge of PL/I.

---

[1]NPS LISP is the name given to this version of LISP 1.5 implemented at the Naval Postgraduate School.

II. DISCUSSION OF LISP IMPLEMENTATIONS

Five existing LISP systems were studied prior to beginning the system design of NPS LISP. The systems examined were 7090 LISP, PDP-1 LISP, Q-32 LISP, M-460 LISP, and BBN 940 LISP. Their designations are derived from the type of computer on which each system was implemented. During the study of each LISP system, particular attention was given to the internal data structure, the method of maintaining variable values (bindings), and other features relevant to an interpreter-oriented LISP system. Those features related to a particular system's LISP compiler were not examined closely, nor are they discussed in this section.

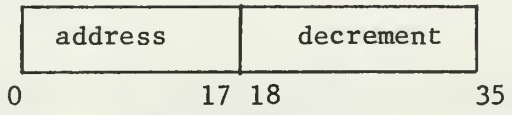
The discussion of each LISP system in this section includes some general remarks about the system, followed by a description of the internal data structure and the variable binding methods.

A. 7090 LISP

7090 LISP is a complete and powerful system that was implemented on the IBM 7090 at M.I.T. by a group that included John McCarthy, the designer of the first LISP programming system [Refs. 1 and 2]. It is used as a basis for comparing the other LISP systems discussed in this section. The 7090 LISP system accepts as input a pair of S-expressions on punched cards which are evaluated by the function EVALQUOTE. The first S-expression is a function name or definition. The second S-expression is a list of arguments of the function.

1. Data Structure

LISP programs and data are represented in memory by lists. Each element of a list in 7090 LISP is represented by a cell which is one 36-bit computer word. The cell is divided into two 18-bit fields and is graphically represented as follows:



The left and right portions of the cell are referred to as the CAR (contents of the address register) and the CDR (contents of the decrement register), respectively.

An atomic symbol, or atom, is represented in memory by a special type of cell called an atom header cell followed by a property list. Atom header cells are characterized by the presence of the constant -1 in the CAR of the cell. The CDR of an atom header cell points to the atom's property list. In 7090 LISP, property lists contain both information required for system operation and user-defined properties. Atom print names are stored in a non-list area of memory called full word storage. Six characters of a print name are packed in a full word. The print name full words are accessed by a list of pointers on the property list. Figure 1 is a representation of the structure of the atom LISTING which is a function defined by the S-expression (LAMBDA (X) X).

A numerical atom, or number, is represented in 7090 LISP by an atom header cell whose CDR points to a full word containing the value of the number. 7090 LISP accepts and operates on both fixed-point

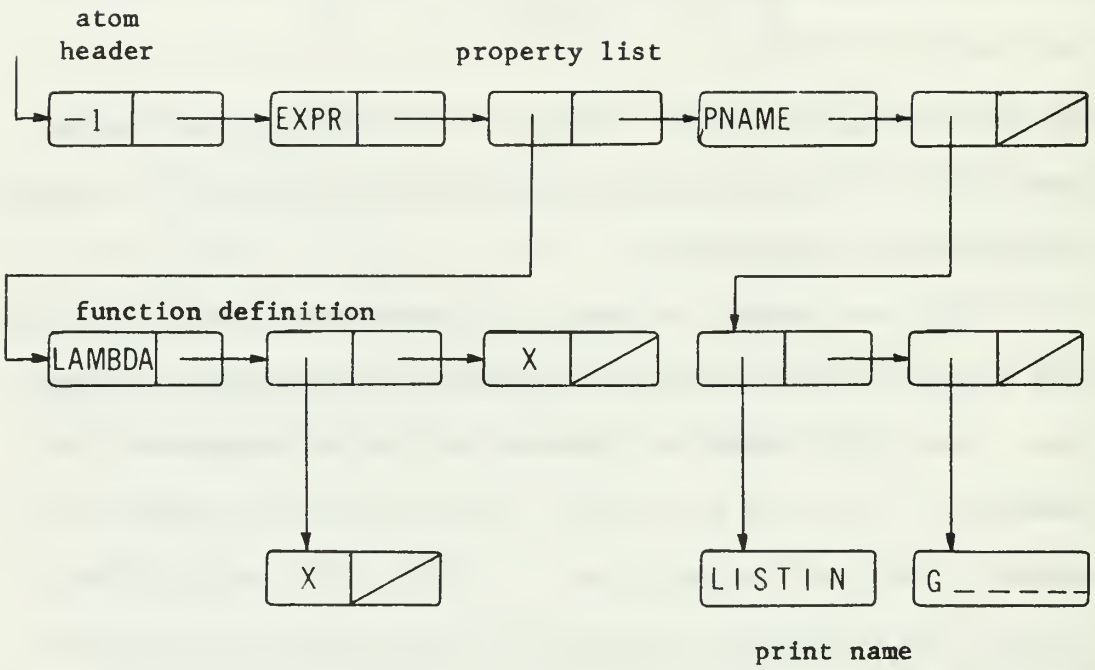
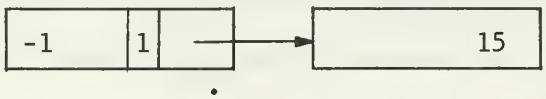


FIGURE 1

THE 7090 LISP FUNCTION LISTING DEFINED BY (LAMBDA (X) X)

(integers) and floating point numbers. A tag field in the atom header cell specifies the type of number represented. For example, the 7090 LISP representation of the integer 15 is diagrammed below:



2. Variable Bindings

Function definitions in LISP contain the special form LAMBDA as the first element in the S-expression definition. In 7090 LISP, LAMBDA causes the dummy variables in the function to be bound to their corresponding arguments on an association list (A-list). The association list is a list of pairs. Each pair consists of a dummy variable and its corresponding argument.

Suppose, for example, that the following pair of S-expressions were input to the system:

(LAMBDA (X) (CDR X)) ((A B C))

The interpreter would recognize LAMBDA and then bind the dummy variable X to the list (A B C) on the A-list. Thus, the A-list would be ((X . (A B C)). When the form (CDR X) is subsequently evaluated, the system retrieves (A B C) as the value of X, applies the function CDR, and returns (B C).

Permanent bindings of variables, called zero-level bindings, are made on property lists with the indicator APVAL. For example, if the atom X has a permanent value of (A B C), its property list would appear as illustrated in Figure 2.

B. PDP-1 LISP

PDP-1 LISP was developed by L. Peter Deutsch for the PDP-1 computer made by the Digital Equipment Corporation [Ref. 1]. PDP-1 LISP is a small flexible system incorporating only a limited number of LISP functions. However, the system has two significant features. First, it can operate in as few as 2000 words of core memory. Secondly, it allows correction of S-expressions previously input to the system. PDP-1 LISP accepts a single S-expression for evaluation, input on punched paper tape or through a teletype keyboard. The S-expression is evaluated by the function EVAL.

1. Data Structure

A LISP cell in PDP-1 LISP is formed from two contiguous 18-bit words. Four bits in each word are available for use as tags since only 14 bits are required for addressing list structure. PDP-1 LISP uses the LISP cell for storing all types of data; it does not have a full word storage area.

The structure of PDP-1 LISP atom property lists is essentially the same as 7090 LISP property lists. Differences are found in the structure of the atom header cells and the manner in which print names are stored. The CAR of the atom header cell points to the print name while the CDR points to the property list. Print names are stored in list structure with three characters packed in the CAR of each list element. Figure 3 shows the structure of the atom LISTING, where LISTING is a function defined by the S-expression (LAMBDA (U) U).

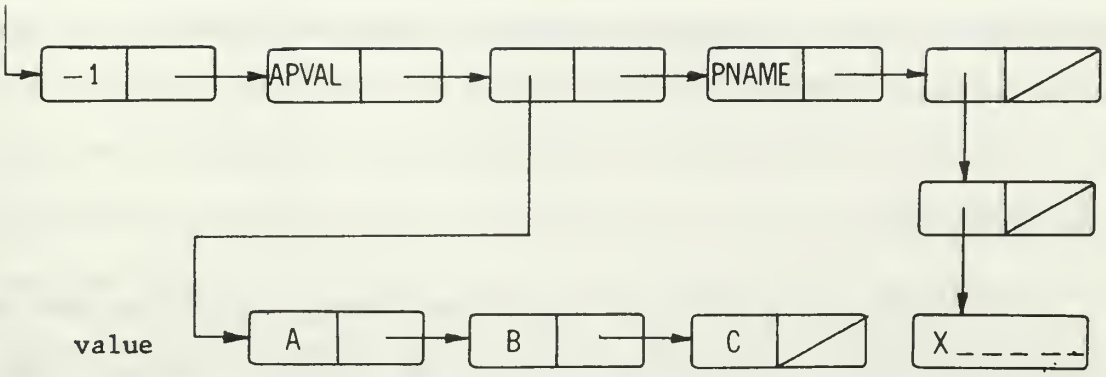


FIGURE 2

EXAMPLE OF A 7090 LISP ATOM WITH A PERMANENT VALUE

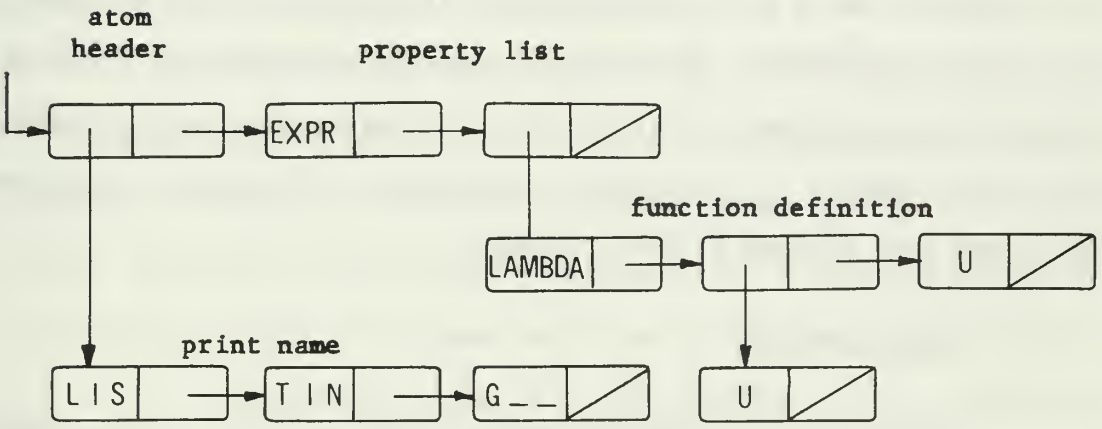


FIGURE 3

EXAMPLE OF A PDP-1 LISP FUNCTION DEFINED BY (LAMBDA (U) U)

2. Variable Bindings

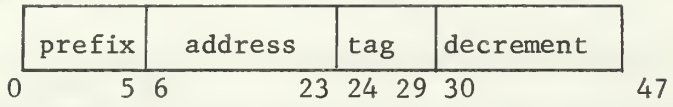
PDP-1 LISP variable binding techniques are similar to those of 7090 LISP. That is, LAMBDA causes dummy variables to be bound on an association list. Permanent values are placed on property lists after the indicator APVAL.

C. Q-32 LISP

Q-32 LISP was developed at System Development Corporation for operation under SDC's time-sharing system on the AN/FSQ-32 computer [Ref. 3]. Q-32 LISP is a very extensive system that incorporates all but a few of the features of 7090 LISP. A principal feature of Q-32 LISP is that it is completely compiler-oriented. Thus, all functions are compiled before execution. The system evaluates pairs of S-expressions input through a remote teletype terminal. As in 7090 LISP, the pair consists of a function and a list of arguments. If a function name is input as the first S-expression, the compiled code which defines the function is applied to the arguments. If the first S-expression contains LAMBDA as its first element, it is a function definition. Therefore, it is compiled and then applied to the arguments.

1. Data Structure

One 48-bit Q-32 computer word, divided into four parts, forms a LISP cell for storing list structure. The prefix, decrement, tag, and address occupy bits 0-5, 6-23, 24-29, and 30-47, respectively. It should be noted that in other LISP systems discussed here, the decrement is contained in the right part of the LISP cell and the address is in the left part. For consistency, the Q-32 LISP cell will be shown as follows:



Atom property list structures in Q-32 LISP differ considerably in complexity and form from 7090 LISP property lists. A primary reason for this difference is that Q-32 LISP is compiler-oriented rather than interpreter-oriented. The atom header cell of a literal atom contains an atom flag in the prefix (bit 1 = 1), along with a CDR pointer to its print name and property list. The CAR of the atom header cell is either NIL or points to a special binding which is similar to an APVAL binding in 7090 LISP. The CAR of the first cell on a property list following the atom header cell points to an array containing the print name. The remainder of the property list is used only for storing properties defined by the LISP programmer. The 7090 LISP indicators PNAME, APVAL, SUBR, and EXPR are not used in Q-32 LISP.

Q-32 LISP also utilizes quote cells. The quote cells are used to designate atoms and active list structure during garbage collection. The CAR of a quote cell points to an atom header cell or a list structure. The CDR of a quote cell is always NIL. A reserved region of core memory contains the quote cells and literal atom header cells.

As an example, consider the list structure for a typical atom in Q-32 LISP. Referring to Figure 4, assume that the atom ATOMLISTA has a zero-level binding to the list (A B C) and that the property J has been put on the property list under the indicator PROP. The print name of ATOMLISTA is stored in an array in full word storage. The first word in the array indicates that the array contains a print name (octal 03 in the prefix). The first word also indicates that the print

name uses two words of storage (2 in the CDR) and that there is one character stored in the last word of the array (1 in the tag).

The atom header cell of a function name has a CAR pointer to the first cell of compiled code for the function. For example, the structure of the LISP function EQ is shown in Figure 5.

The storage of numbers is similar to print names. Numerical atom header cells are stored in free storage instead of in the reserved area of memory with literal atom header cells. For example, the number 101 is represented by the structure given in Figure 6. The atom header cell of the numerical atom 101 contains 71 in the tag field to indicate the number is an integer. The CAR of the atom header cell points to an array in full word storage containing the value of the number. The array header contains octal 02 in the prefix indicating a numerical array. The number 1 in the CDR specifies that one data word is used. Note that the arrays in full word storage are sets of contiguous memory locations.

The OBLIST is the structure which allows literal atoms to be represented uniquely in Q-32 LISP. The OBLIST is a set of 125 contiguous words in memory which point to a list of pointers to literal atoms. It is used by the read routines as a rapid look-up table for atoms to determine if they are present in memory. A simple hash coding scheme based on the first eight characters of the print name is used to determine the placement of an atom in the OBLIST.

2. Variable Bindings

Q-32 LISP does not use an association list for maintaining variable values bound by LAMBDA. Instead, bindings of dummy variables are kept in a pushdown list. Referring to Figure 4, note that zero-level bindings are referenced by a CAR pointer in the atom header cell.

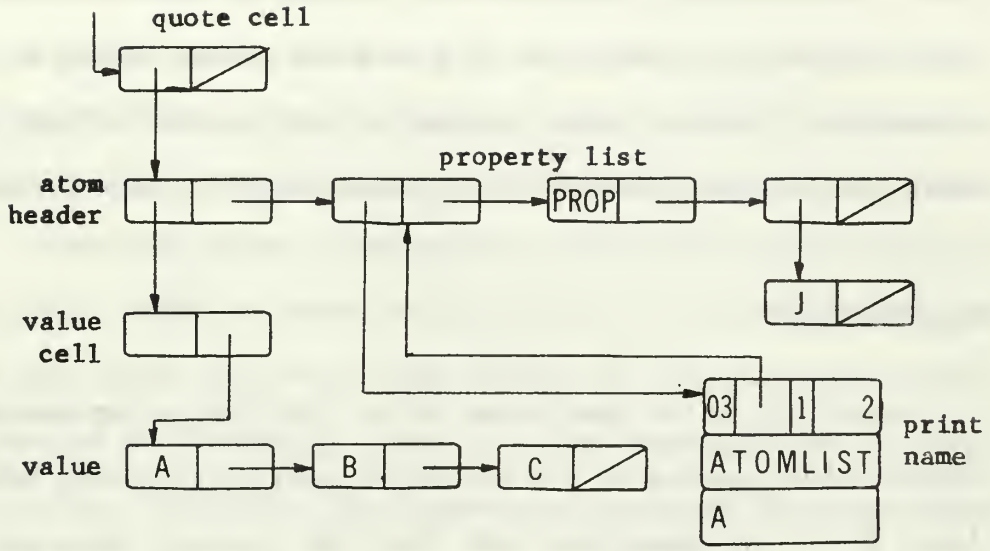


FIGURE 4

EXAMPLE OF A Q-32 LISP ATOM WITH A
PERMANENT VALUE

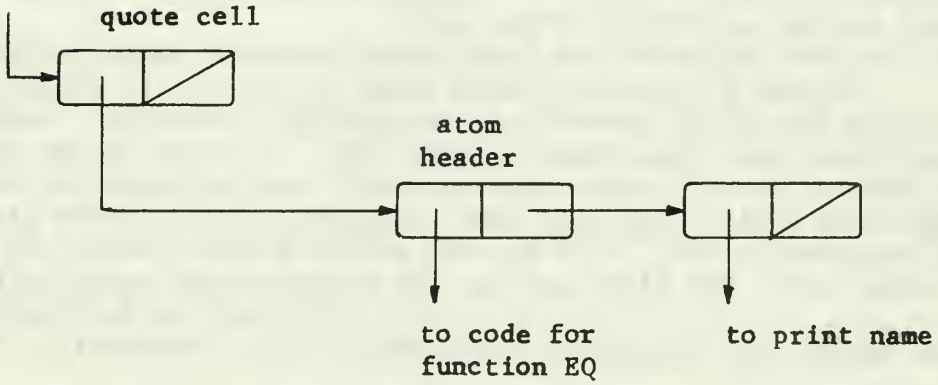


FIGURE 5

STRUCTURE OF THE Q-32 LISP FUNCTION EQ

This discussion of variable bindings has been greatly simplified. In actuality, there are other aspects of variable bindings in Q-32 LISP which are important if a complete understanding of Q-32 LISP is desired. However, these aspects are not covered in this section since they are not essential in an interpreter-oriented system.

D. M-460 LISP

M-460 LISP is the name given to the LISP system implemented on the UNIVAC M-460 computer at the Air Force Cambridge Research Laboratories [Ref. 1]. It is compatible with 7090 LISP although there are some features not implemented.

1. Data Structure

The M-460 LISP cell consists of one 30-bit M-460 computer word. Fifteen bit fields are used for both the address and decrement values. There are no tag fields in the cell.

Property lists of literal atoms in M-460 LISP differ in several significant ways from those in 7090 LISP. In place of the 7090 LISP indicators APVAL, SUBR, and EXPR, property lists in M-460 LISP contain a value cell. The first cell on the property list which follows the atom header cell contains a CAR pointer to the value cell. The CAR of the value cell points to one of the following: (1) a zero-level binding; (2) the code for a compiled function; or (3) a LAMBDA expression which defines a function. In the case of a compiled function, the value cell also contains a CDR pointer to a SETUP routine which stores parameters and variable values on the pushdown list.

The LISP function CAR is an example of a compiled function. The structure of the property list of CAR is shown in Figure 7. The function FN defined by the S-expression (LAMBDA (Y) (PRINT Y)) has the structure illustrated in Figure 8. Note that the print name is stored in a list with one character in the CAR of each list element.

Numerical atoms are structured in a rather unique fashion in M-460 LISP. Numerical atoms are stored in a list structure composed of an atom header cell with a CDR pointer to a list of number cells. The value of the number is stored in a list containing one to three number cells. The CAR of each number cell contains ten significant bits and a sign bit. The CDR points to any remaining number cells. Note that there is no full word storage in M-460 LISP memory.

2. Variable Bindings

In contrast to 7090 LISP, the M-460 LISP system does not have an association list for maintaining values of variables. Instead, bindings of dummy variables established by LAMBDA are kept on a push-down list. Zero-level bindings are referenced by the CAR of the value cell on the property list. Suppose, for example, that the atom ATM has a zero-level binding to the list (G H I). The structure of ATM is represented in Figure 9.

E. BBN 940 LISP

BBN 940 LISP is an upward compatible extension of 7090 LISP implemented on the SDS 940 computer by Bolt Beranek and Newman, Inc. [Refs. 4 and 5]. BBN LISP includes a number of features which give

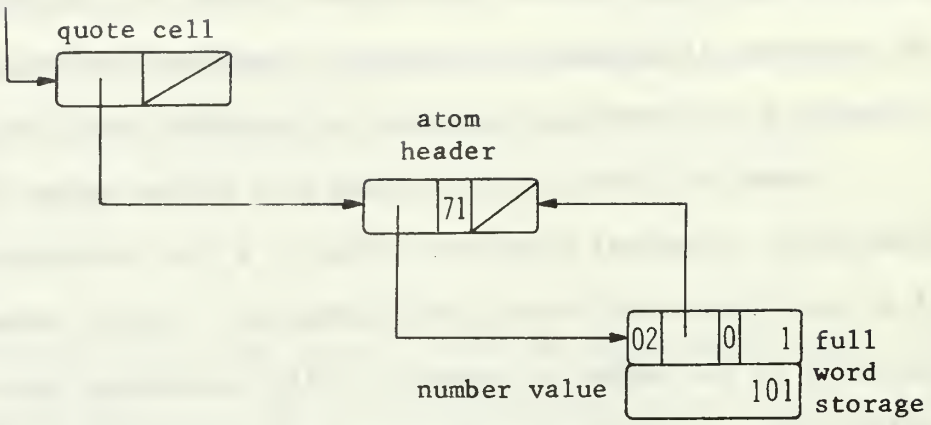


FIGURE 6

STRUCTURE OF A Q-32 LISP NUMBER

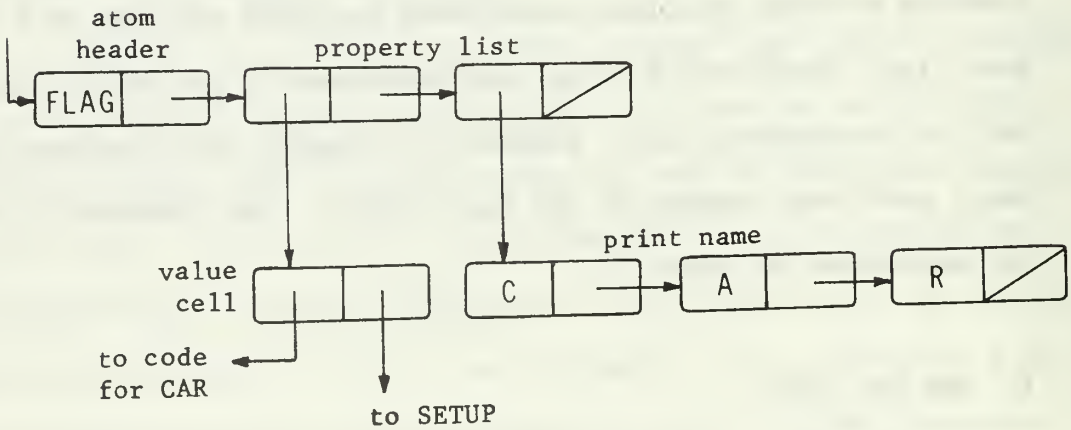


FIGURE 7

STRUCTURE OF THE M-460 LISP FUNCTION CAR

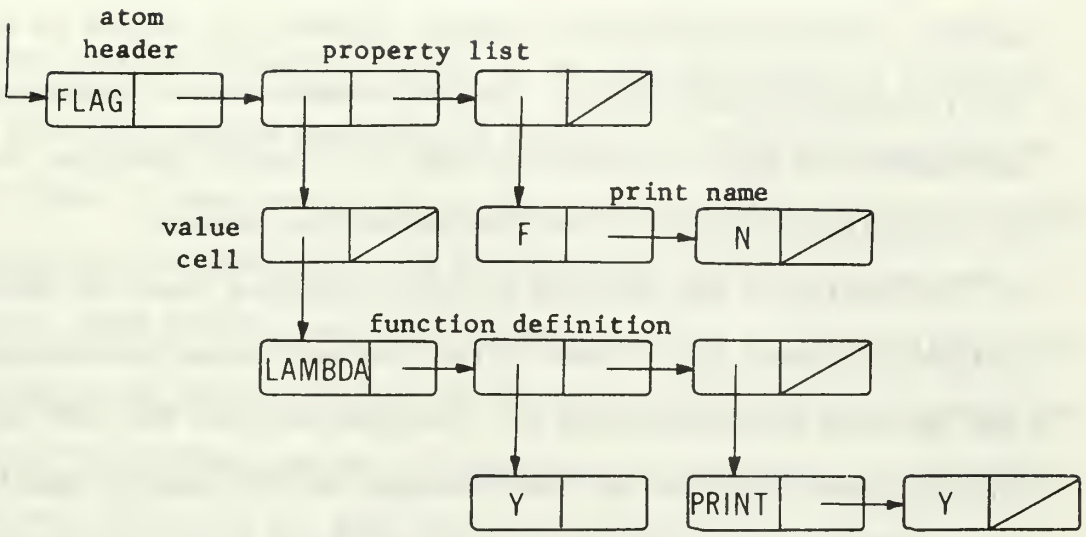


FIGURE 8

STRUCTURE OF THE M-460 LISP FUNCTION FN
DEFINED BY (LAMBDA (Y) (PRINT Y))

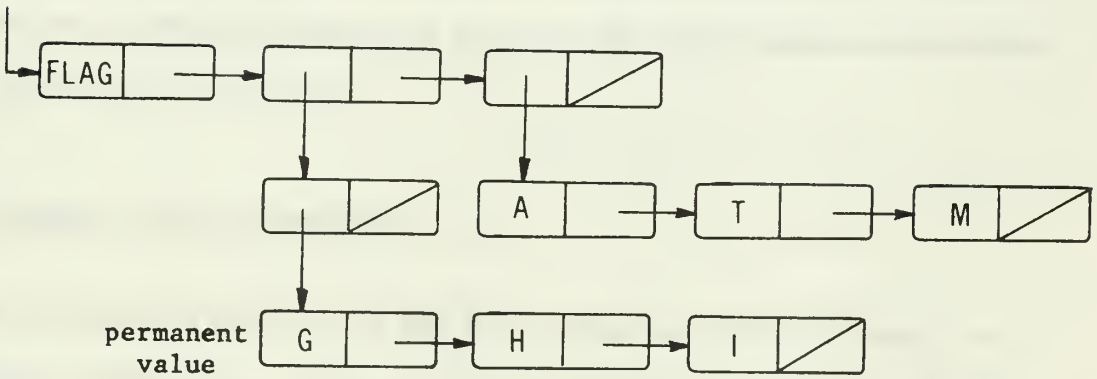


FIGURE 9

EXAMPLE OF AN M-460 LISP ATOM WITH A PERMANENT VALUE

it extensive on-line capabilities. These features include conditional break-points in functions for debugging along with a sophisticated LISP editor. The most significant feature, however, is the use of a drum as the principal storage device. Careful segmentation of system code, arrangement of data in memory by type, and special attention to binding of variables contribute to the success of the system.

The design of BBN 940 LISP is quite different from the other LISP systems discussed here. None of the internal system conventions of BBN 940 LISP were considered for incorporation into NPS LISP because of the system's use of two level storage and its complex compiler-oriented structure. BBN 940 LISP is mentioned here only because of its unique features and is not further discussed.

The study of these various LISP systems provided many specific implementation ideas for the design of NPS LISP. Features borrowed from these LISP systems for NPS LISP are described in the next section.

III. NPS LISP SYSTEM DESIGN AND IMPLEMENTATION

The design and implementation of NPS LISP was based on a study of the five LISP systems described in Section II. During the course of this study, it was found that no LISP system in particular was decidedly superior to the others from the standpoint of internal system design. However, 7090 LISP is described in greater detail in the available literature and it has the distinction of being a development of the original LISP programming system. It was decided, therefore, to model NPS LISP after 7090 LISP for both technical and aesthetic reasons.

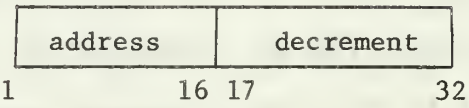
The discussion of the design and implementation of NPS LISP in this section begins with a description of the internal system conventions adopted for list representation, atom property lists, character objects, numerical atoms, the OBLIST, and memory organization. Next, is a complete and detailed description of the NPS LISP interpreter. A short discussion of NPS LISP functions and some of their common characteristics concludes this section.

A. INTERNAL SYSTEM CONVENTIONS

The internal structure of NPS LISP resembles 7090 LISP more than any other LISP system. However, design features of other systems were incorporated into NPS LISP when they appeared to be advantageous. It was desired to keep the NPS LISP system structure uncomplicated, yet allow for the basic capabilities of 7090 LISP. Thus, the internal structure described in this section is a compromise between structural simplicity and system sophistication.

1. List Representation

In NPS LISP, lists are represented in memory by a linked list structure build from 32-bit computer words. Each cell can be graphically represented as follows:



The address and decrement fields of the cell are 16-bit fields whose first bit is reserved for use in garbage collection. The maximum size of NPS LISP memory is limited to the maximum value representable in 15 bits --32,767.

All LISP programs and data, except numbers, are stored in list structure. The system does not rely on values being located in successive memory cells; instead, the interpreter accesses particular values by traversing lists. Since LISP lists are essentially binary trees, the list traversing procedures are relatively straightforward.

2. Atoms and Property Lists

Atomic symbol representation and property lists in NPS LISP correspond closely to those in 7090 LISP. Every atomic symbol is represented uniquely. In addition, each atom has a property list created when the atom is read.

Initially, each atom has only its print name on its property list. A special type of cell, called an atom header cell, is the first cell on a property list. An atom header cell contains -1 in its CAR. The CAR of an atom header cell is the only element of list

structure containing a negative number. Thus, it is a simple matter to determine when an atom has been reached during list traversals by testing for a value less than zero. The CDR of an atom header points to the property list.

Some properties that are on property lists of literal atoms are preceded by special system defined atoms called indicators. The indicators used by NPS LISP and their meanings are:

PNAME	print name of the atom
APVAL	permanent value of an atom considered as a variable
SUBR	function defined by a PL/I procedure
EXPR	function defined by an S-expression
FSUBR	special form defined by a PL/I procedure
FEXPR	special form defined by an S-expression

In addition to the indicators used by the LISP system, the LISP programmer may put other indicators on property lists.

A property list indicator not followed by a property is called a flag. Flags indicate a certain property by their presence alone. An example of a flag is the trace flag, FLAG. FLAG is put on the property list of a function to cause trace information to be printed when the function is executed.

The method of atom print name storage was taken from M-460 LISP. An atom print name is stored in a list. The CAR of each cell in the list contains the EBCDIC code of one character of the print name. The CDR is a pointer to the remainder of the name. For example, the atom ABCD is represented in NPS LISP memory as shown in Figure 10. It is emphasized that while PNAME and NIL in Figure 10 are pointers to the atoms PNAME and NIL respectively, the characters A, B, C, and D are the

actual EBCDIC code for those characters. In Figure 10, the computer word containing the character A appears as follows, assuming B is contained in cell 1024:

A		1024	
00000000	10111111	00000100	00000000

The binary number 10111111 in the second byte above corresponds to decimal 191, the EBCDIC code for the character A. Note that the first byte of a print name cell is unused since the first bit is reserved for garbage collection.

The indicators SUBR and FSUBR on an atom property list signal that the atom is a function defined by a PL/I procedure. Each PL/I coded function is assigned a unique integer number. The interpreter uses this number to transfer control at the time of the function call. For example, the LISP function CAR has the structure diagrammed in Figure 11. The number 2 appearing after SUBR is the unique number assigned to the function CAR.

Functions defined by S-expressions in NPS LISP have one of the indicators EXPR or FEXPR on their property lists. Figure 12 is a diagram of the function LISTING, where LISTING is defined by the S-expression (LAMBDA (X) X).

Literal atoms which have a permanent value (zero-level binding) have the indicator APVAL on their property lists. If, for example, the atom LISTA has a permanent value of (A B C D), its property list is structured as in Figure 13.

There are three literal atoms in NPS LISP which have themselves as values: T, F, and NIL. The atoms T and F can be interpreted as the boolean values true and false. NIL is used to terminate a list.

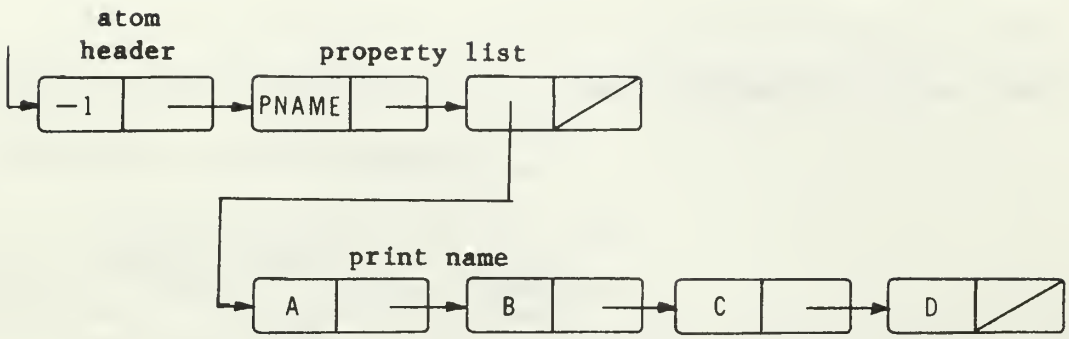


FIGURE 10

STRUCTURE OF AN ATOM IN NPS LISP

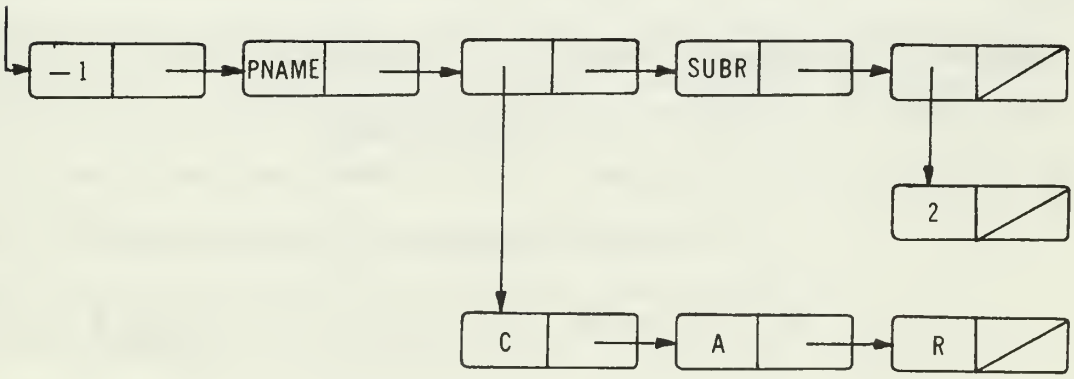


FIGURE 11

EXAMPLE OF A PL/I CODED LISP FUNCTION

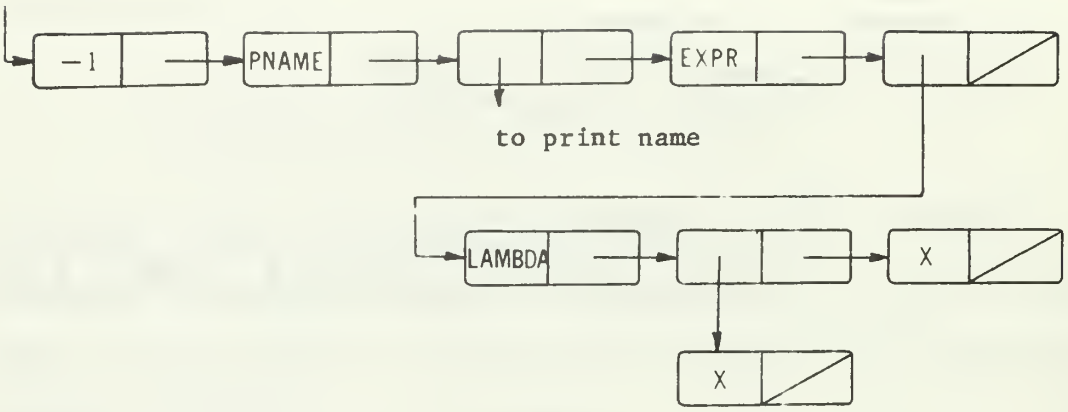


FIGURE 12

EXAMPLE OF AN NPS LISP FUNCTION DEFINED BY (LAMBDA (X) X)

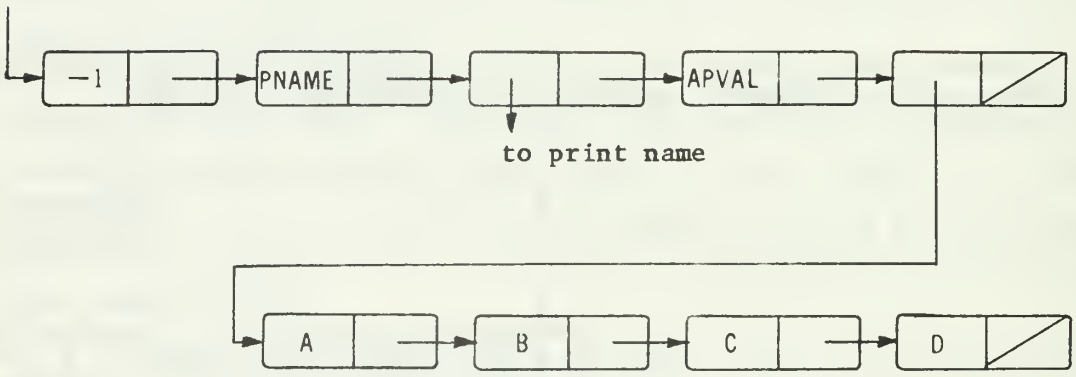


FIGURE 13

EXAMPLE OF AN ATOM IN NPS LISP WITH A ZERO-LEVEL BINDING

The property lists for T, F, and NIL are in Figure 14. Note that in Figure 14 the value of T is T, the value of F is NIL, and the value of NIL is NIL. This may appear confusing, but the interpreter is constructed to evaluate T, F, and NIL to their proper values; the circular lists are not a problem here.

3. Character Objects

NPS LISP handles special characters in a manner similar to 7090 LISP. Special characters are characters other than letters and digits. The legal special characters, called character objects, have print names that are the mnemonic equivalent of their corresponding typewriter character, and values which are their EBCDIC codes. For example, the character object RPAR (right parenthesis) has the structure given in Figure 15.

Character objects are not stored in memory with their atom header cells in successive memory locations. Instead, they are stored as any other literal atom. The unique characteristic of a character object is that the value of the object is an EBCDIC code rather than a pointer to an atom header or list. The legal character objects and their permanent values are listed in Table 1.

4. Numbers

Only integer numbers are allowed in NPS LISP. The manner of storing numbers in NPS LISP is similar to that of 7090 LISP. A number is an atom that does not have a property list. The CDR of the atom header of a numerical atom points to an area of memory called full

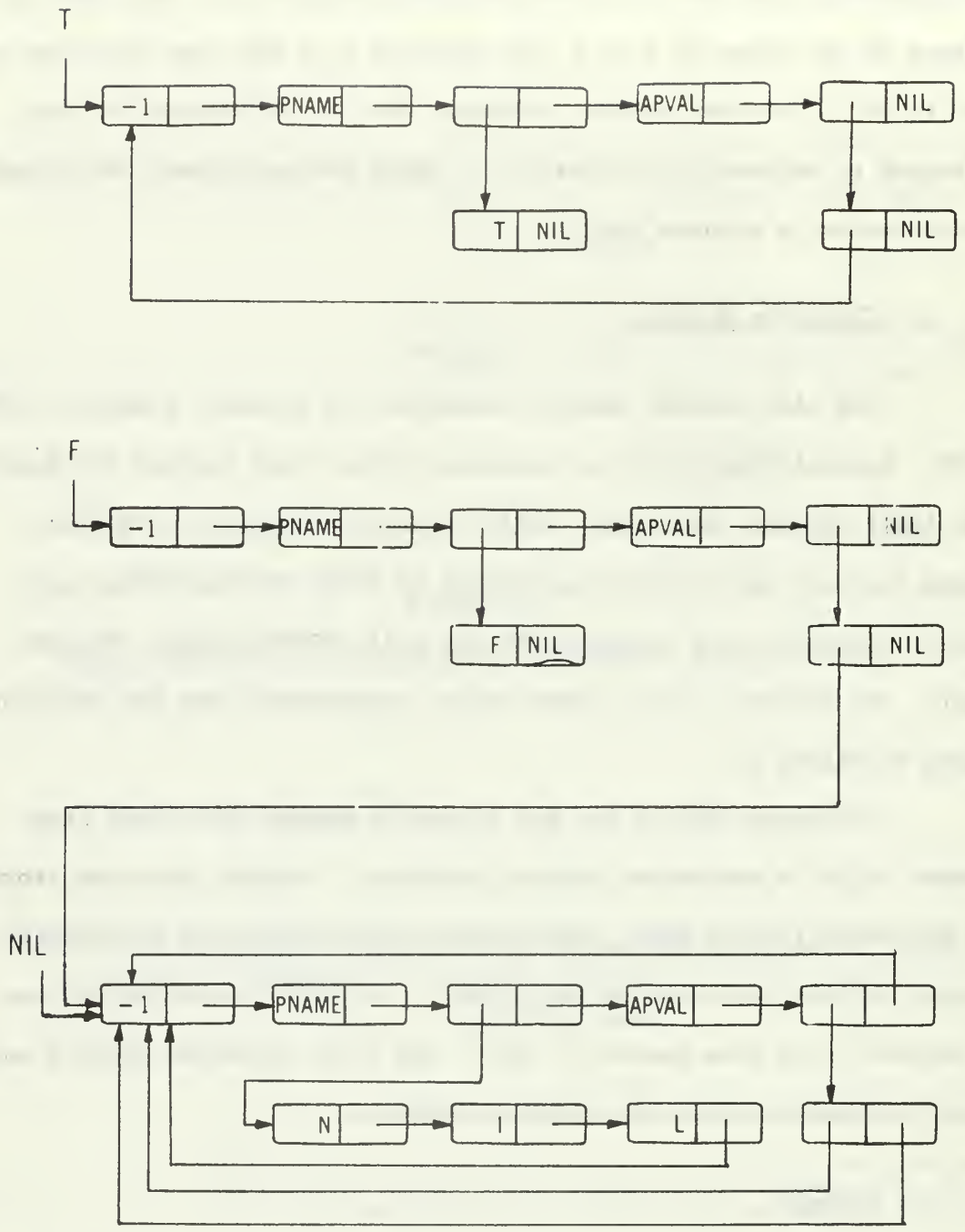


FIGURE 14

STRUCTURE OF THE ATOMS T, F, AND NIL

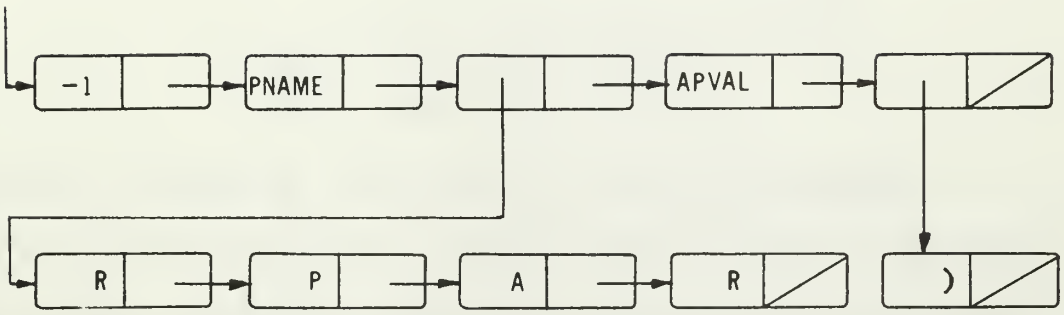


FIGURE 15

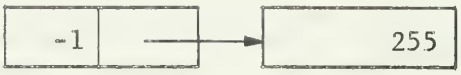
STRUCTURE OF A CHARACTER OBJECT IN NPS LISP

TABLE I

NPS LISP CHARACTER OBJECTS

Print Name	Permanent Value
DOLLAR	$
SLASH	/
LPAR	(
RPAR	)
COMMA	,
PERIOD	.
PLUS	+
DASH	-
STAR	*
BLANK	blank
EQSIGN	=

word storage. The value of the number is stored in full word storage as a signed binary integer. For example, the number 255 has the following representation:



Numbers are not unique in NPS LISP. For example, each occurrence of the number 5 in the list (1 5 2 5) is represented by a separate atom header cell whose CDR points to a different location in full word storage. Figure 16 is a diagram of the list (1 5 2 5) illustrating the method of number representation.

5. The OBLIST

The object list, or OBLIST, is the structure which permits literal atoms to be represented uniquely in memory. The OBLIST consists of 127 contiguous memory cells each containing a CDR pointer to a sub-list. The CAR of each element on a sub-list points to an atom that has either been read or is part of the initialized system. When atoms are read, a hash code is computed based on the first three characters of the atom print name and the length of the print name. An atom is placed in the OBLIST according to its hash code -- a number between one and 127. The CAR of each cell in the OBLIST contains the number of atoms that exist in the system and which hash code to the same OBLIST location.

For example, suppose the atoms FORM1 and FORM2 have a computed hash code of 57. If they are the only atoms in the system with this particular hash code, the relevant section of the OBLIST and its sub-list is represented in Figure 17.

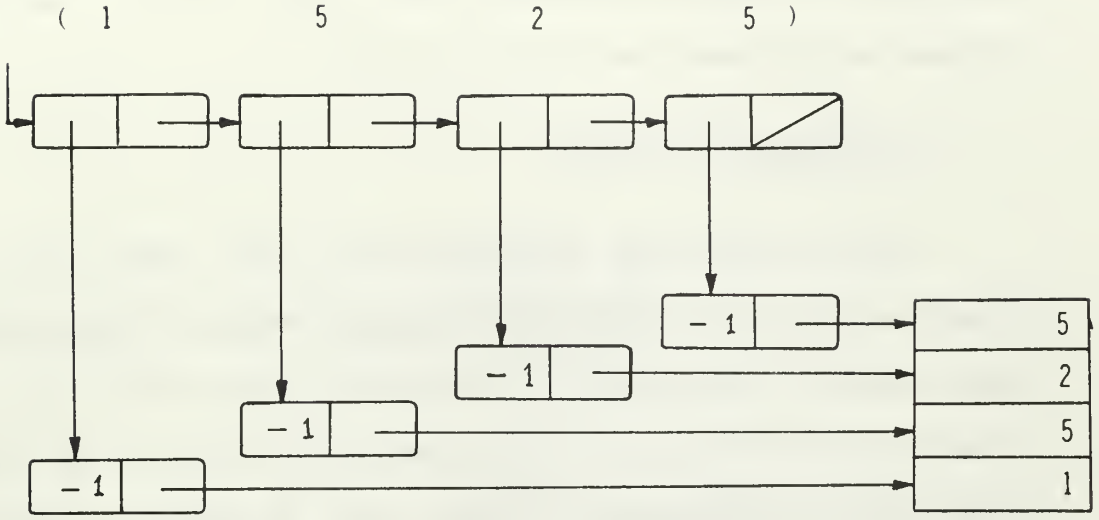


FIGURE 16

EXAMPLE OF NON-UNIQUE NUMBER STORAGE

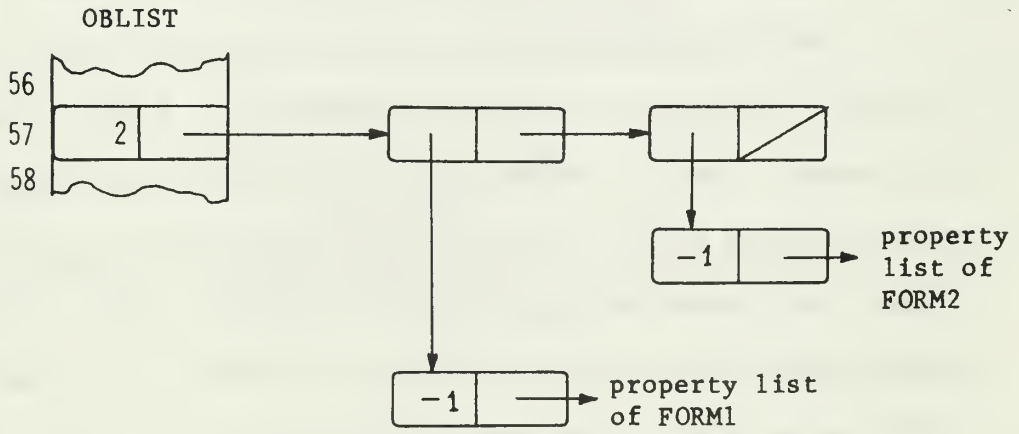


FIGURE 17

EXAMPLE OF THE OBLIST STRUCTURE

The read routines ensure that multiple references to the same atom point to the same atom header cell. Figure 18 shows a multiple reference to the atom E in the list (E E F).

6. Organization of Memory

Since the NPS LISP system was coded in PL/I, LISP memory is actually an array named FSTOR (free storage) having the attributes BINARY FIXED(31) and a dimension of (0:32766). Each LISP cell is one element in the FSTOR array. The CAR and CDR of each cell are either indexes into the FSTOR array or EBCDIC codes for the print name characters or character objects.

NPS LISP differs from other LISP systems discussed in one important way: all NPS LISP system functions including the interpreter and I/O routines are written in PL/I. Thus, most of the 32K LISP memory is available for storing list structure and numbers.

The OBLIST was placed in the lowest 127 cells of LISP memory in order to maintain a reasonable resemblance to a more conventional LISP system. That is, FSTOR(1:127) corresponds to the OBLIST. The atom structures for all built-in LISP functions and legal character objects are created during system initialization. These structures occupy approximately the next 1000 cells in FSTOR. The region of FSTOR beginning at (approximately) FSTOR(1100) to the lower limit of full word storage is allocated for list structure and is called free storage.

Numbers are stored in full word storage located at the top of memory. One cell, FSTOR(32766), is initially allocated for storing

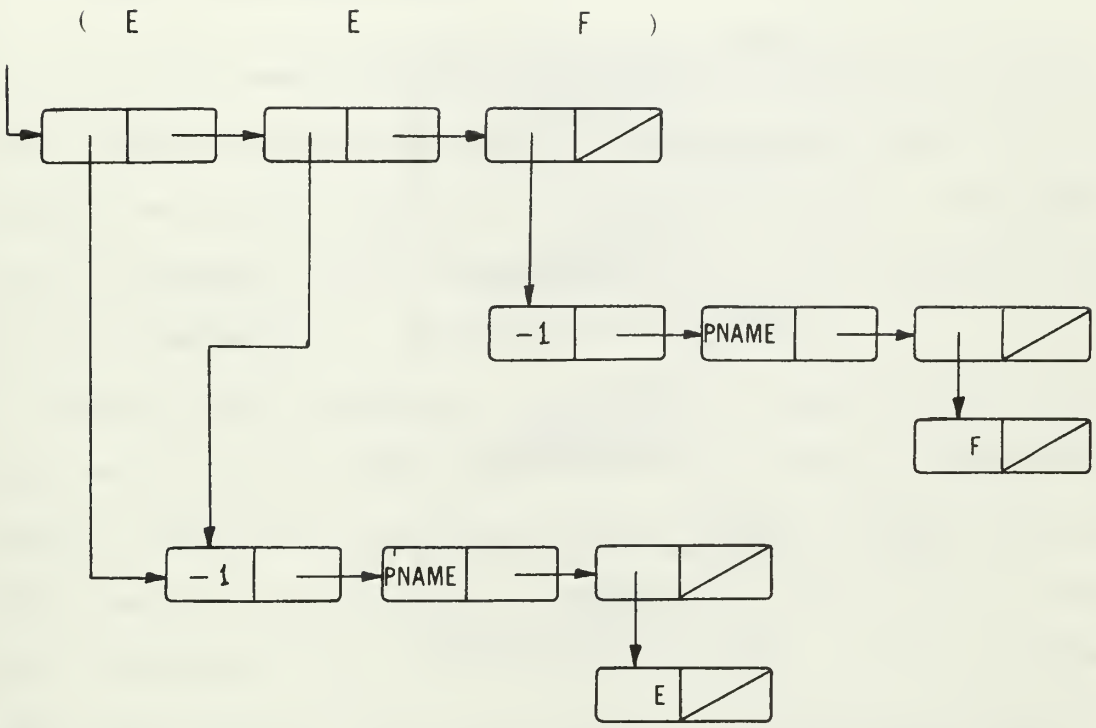
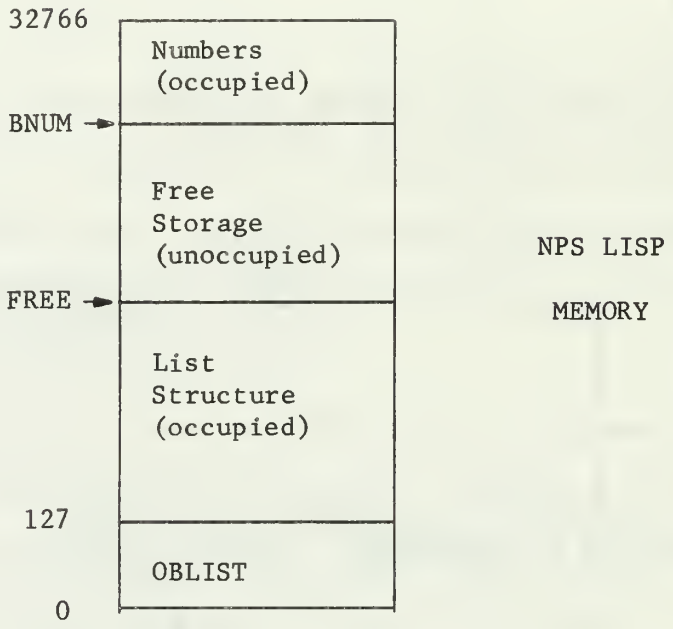


FIGURE 18

ILLUSTRATION OF UNIQUE REPRESENTATION OF LITERAL ATOMS

the first number. Subsequent numbers are placed in FSTOR at lower addresses. During the execution of LISP programs, the areas occupied by list structure and numbers grow toward each other. The organization of NPS LISP memory is illustrated below:



FREE and BNUM are PL/I variables which point to the next available cell in free storage and full word storage respectively.

Another departure from the structure of other LISP systems is that NPS LISP does not maintain a pushdown list. Recursive LISP functions are written as recursive PL/I procedures. Thus, partial results of computations are maintained automatically. S-expressions which define recursive functions are evaluated by the recursive PL/I procedures NAPPLY and NEVAL. Difficulties may occur in the implementation of the garbage collection routines due to the exclusion of the pushdown list. Garbage collection is discussed further in Section IV.

Once the internal system conventions of NPS LISP were established, a few elementary LISP functions were written and tested to

check the validity of the system structure. The interpreter that was written, based on the design conventions, is described in the following paragraphs.

B. THE NPS LISP INTERPRETER

A thorough knowledge of how the interpreter operates is essential to a complete understanding of any interpretive LISP system. For this reason, the NPS LISP interpreter is described in detail. The reader may wish to refer to the listing of the interpreter in Appendix C while reading this section.

The NPS LISP interpreter, a PL/I procedure called NEVALQ, consists of a set of PL/I procedures which perform the essential tasks of evaluating S-expressions input to the system. NEVALQ is called by the NPS LISP supervisor, LISPA, after two S-expressions have been read. These two S-expressions are the two arguments of NEVALQ. The first argument is a function name or function definition; the second argument is a list of arguments for the function. The interpreter is patterned after the 7090 LISP interpreter and contains the same 7090 LISP functions EVALQUOTE, EVAL, APPLY, EVLIS, and EVCON. In addition, the interpreter contains procedures required by NPS LISP due to the structure of the system. Each of the major interpreter functions is discussed separately below. The first function considered is a PL/I procedure vital to the operation of NPS LISP.

1. The Procedure NPROC

NPROC is a PL/I procedure that provides the interface between the interpreter and the LISP functions coded as PL/I procedures.

As mentioned in Section III, each LISP function coded in PL/I has either SUBR or FSUBR followed by a unique number on its property list. This number is used by NPROC to transfer to the subscripted label variable A. The statement in NPROC following A(i) is a call to the corresponding LISP function.

As an example, refer back to Figure 11 where the structure of the function CAR is diagrammed. The number 2 following SUBR on CAR's property list is the number used by NPROC to transfer to the statement with the label A(2). NPROC also assigns the arguments of LISP functions to the standard PL/I variables ARG(1), ARG(2), and ARG(3). Note that this restricts only PL/I coded LISP functions to three arguments. The value of NPROC is the value of the LISP function called by NPROC.

2. EVALQUOTE

The first few executable statements of NEVALQ, although not written as a separate procedure, serve the same purpose as the 7090 LISP function EVALQUOTE. These statements determine if the first argument of NEVALQ is a function with either FEXPR or FSUBR on its property list. If so, the function name is paired with the arguments and EVAL is called with a NIL association list. Otherwise, APPLY is called with a NIL association list (A-list).

3. APPLY

The three primary tasks of APPLY are to (1) apply a function to its arguments; (2) bind variables and function names on the A-list by evaluating the special forms LAMBDA and LABEL; and (3) evaluate functional arguments by detecting the special form FUNARG. The three

arguments of APPLY are a function name or definition, a list of arguments for the function, and the current A-list. Each of the principal tasks of APPLY are examined below in the order that they are accomplished.

Suppose a function name is supplied as the first argument of APPLY. First, the property list of the function is searched for a function definition. The indicator SUBR appearing on the property list means that the function definition is a PL/I procedure and causes control to pass to NPROC. The procedure NPROC then calls the proper LISP function. If the function has an S-expression definition as indicated by EXPR, APPLY is called recursively with the function definition in place of the function name.

Assuming that neither SUBR or EXPR is found on the property list of the function name, APPLY searches the A-list for a function definition. If one is found, APPLY is called recursively with the function definition replacing the function name. If no function definition is found on the A-list or property list, the function is undefined and an error message is printed.

The first element of a function supplied to APPLY may be one of the special forms LAMBDA, LABEL, or FUNARG. The appearance of LAMBDA initiates two actions. The list of dummy (local) variables following LAMBDA are bound on the A-list to their corresponding arguments by the LISP function PAIR. Further, the function definition is evaluated by EVAL using the modified A-list. When the special form LABEL is encountered, the name appearing after LABEL is bound to its function definition on the A-list. APPLY is then called with the function definition as its first argument. For example, the S-expression (LABEL FN (LAMBDA (X) (CDR X))) adds (FN . (LAMBDA (X)

(CDR X))) to the A-list and APPLY is called recursively with (LAMBDA (X) (CDR X)) as its first argument. In most LISP systems, FUNARG is referred to as a device rather than a special form since it is used only within the system. When a function has a functional argument, FUNARG ensures the correct retrieval of bindings of free variables which occur in the functional argument. An excellent discussion of the necessity and operation of the FUNARG device is contained in Ref. 1, pages 63-65.

It may happen that the first argument of APPLY is not in a form suitable for evaluation by APPLY. In this case, EVAL evaluates the first argument and then APPLY is called recursively with the value of EVAL as its first argument.

4. EVAL

The primary purpose of EVAL is to evaluate forms. A form is defined as an S-expression consisting of (1) a numerical or literal atom; (2) one of the special forms QUOTE, FUNCTION, or COND, followed by its arguments; or (3) a function followed by its arguments. EVAL requires two arguments -- a form to be evaluated and the current A-list. EVAL returns an S-expression that is the value of the form. The actions taken by EVAL in evaluating a form are described below in the order which they occur.

The first argument of EVAL may be a numerical or literal atom. If it is a number, as occurs in the S-expression (CONS 1 2), then EVAL merely returns the number as its value. If the first argument is a literal atom, it must have been previously bound to a value on either its property list or the A-list. EVAL first searches the property list of the atom for a zero-level binding under the indicator APVAL.

It returns this value if the search is successful. If no APVAL binding is found, the A-list is searched from the top for the most recent value binding of the atom. If no binding is discovered, an error results and an error message is printed.

The order in which atom values are retrieved is important. Since property lists are searched before the A-list for value bindings, a zero-level binding always takes precedence over any bindings that are on the A-list. The example below illustrates this fact:

CSET (A (1 2 3))

(LAMBDA (A B) (CONS A B)) (X Y) = ((1 2 3). Y)

The atom A is given a zero-level binding to the list (1 2 3) by the function CSET. When the interpreter recognizes LAMBDA, A and B are bound to X and Y on the A-list. As the arguments of CONS are evaluated by EVAL, (1 2 3) is retrieved from the property list of A, and Y is obtained as the value of B from the A-list. Thus, the arguments of CONS are (1 2 3) and Y. The S-expression ((1 2 3) . Y) is returned as the value of CONS.

If the form to be evaluated is not an atom, then it is an S-expression whose first atomic element is a special form or function name. The special forms recognized by EVAL are QUOTE, FUNCTION, and COND. The special form QUOTE causes the element following QUOTE to be returned as the value of the S-expression. For example, the list (A B C D) is returned as the value of (QUOTE (A B C D)).

FUNCTION is a special form which indicates that the S-expression which follows is a functional argument of a function. A list is returned consisting of FUNARG, the functional argument, and the current A-list.

The special form COND is evaluated by the interpreter function EVCON, described later in this section.

Function names are evaluated after considering special forms. A literal atom is assumed to be a function name if it is not one of the special forms above. Therefore, the function definition must be determined. First, the property list of the function definition is searched for one of the indicators SUBR, FSUBR, EXPR, or FEXPR. If SUBR is found, the function has a PL/I definition. Thus, the function is evaluated by invoking NPROC after the arguments of the function are evaluated by EVLIS. FSUBR indicates that the function arguments must be passed directly to NPROC. An S-expression function definition is indicated by the presence of EXPR on the property list. Consequently, the function is evaluated by APPLY after EVLIS evaluates the function arguments. If FEXPR is on the property list, the function arguments are not evaluated. Instead, a list, comprised of the function arguments and the current association list, replaces the arguments of the function. APPLY is then called to evaluate the function.

Finally, if the first element (the CAR) of the form supplied to EVAL is not an atom, then the remainder (the CDR) of the form is evaluated by EVLIS. APPLY is then called to evaluate the first element of the form.

5. EVLIS

EVLIS merely evaluates successive elements of a list by calling EVAL, returning a list of the computed values. EVLIS is used to evaluate the arguments of functions prior to application of the function.

6. EVCON

EVCON is the function which evaluates the conditional form COND. Each of the arguments of COND is a pair of S-expressions for evaluation. The first S-expression of each pair is called the predicate. EVCON successively evaluates predicates while they evaluate to F or NIL. When a predicate is encountered which evaluates to a value other than F or NIL, the second S-expression of this pair is evaluated. The value of EVCON is the value of this S-expression. If all of the predicates evaluate to F or NIL, the conditional is unsatisfied and an error message is printed.

7. Tracing Functions

It is often desirable to trace the execution of a LISP program either for the purpose of debugging or for simply observing the sequence of calls made by the interpreter. The mechanism which enables a trace is provided by (1) the LISP function TRACE which places the indicator FLAG on the property list of functions; and (2) the interpreter procedure PTRACE which prints the trace information in a readable format.

The interpreter functions are traced in the following manner. When APPLY and EVAL are entered, their property lists are searched for the presence of FLAG. If FLAG is found, PTRACE is called to print the value of the function.

Tracing of other LISP functions is handled somewhat differently than the interpreter functions. When the function definition of a PL/I coded LISP function is determined by either APPLY or EVAL, the property list is searched for FLAG. If FLAG is present, the PL/I

variable TRACE1 is set equal to T. PTRACE is called if TRACE1 equals T when NPROC is entered. APPLY and EVAL call PTRACE when FLAG is discovered on the property list of a function defined by an S-expression. An example of the information printed by trace is contained in Figure 19.

8. Error Handling

The NPS LISP interpreter contains no sophisticated error corrections routines and produces few error diagnostic messages. When an error is discovered by the interpreter, a short diagnostic message is printed and NIL is returned as the value of the function currently executing. The interpreter continues evaluating as if NIL is the correct value. Thus, the interpreter attempts to evaluate an S-expression completely even though an error occurs during execution. A LISP error occasionally causes an IBM 360 execution interrupt. As a result, the LISP system must be completely initialized and any functions defined by the LISP programmer are lost. The five errors detected by the interpreter are listed and explained in Table II.

9. Example of Interpreter Execution

Figure 19 is a trace of the steps taken by the interpreter in evaluating the function (LAMBDA (A B) (CONS A B)) applied to the arguments ((X Y Z) (1 2 3)). The arguments to EVALQUOTE were typed in by the LISP programmer; all other printed information was typed by the NPS LISP system. The following sequence of events takes place:

(1) The LISP programmer communicates directly with EVALQUOTE; therefore, he types in two S-expressions -- a function and a list of arguments.

```
CALL EVALQUOTE, ARGS:
_(LAMBDA (A B) (CONS A B)) ((X Y Z) (1 2 3))
CALL APPLY, ARGS:
(LAMBDA (A B) (CONS A B))
((X Y Z) (1 2 3))
NIL
     CALL EVAL, ARGS:
     (CONS A B)
     ((A X Y Z) (B 1 2 3))
          CALL EVAL, ARGS:
          A
          ((A X Y Z) (B 1 2 3))
          VALUE OF EVAL IS:
          (X Y Z)
          CALL EVAL, ARGS:
          B
          ((A X Y Z) (B 1 2 3))
          VALUE OF EVAL IS:
          (1 2 3)
          CALL CONS, ARGS:
          (X Y Z)
          (1 2 3)
          VALUE OF CONS IS:
          ((X Y Z) 1 2 3)
     VALUE OF EVAL IS:
     ((X Y Z) 1 2 3)
VALUE OF APPLY IS:
((X Y Z) 1 2 3)

VALUE IS:
((X Y Z) 1 2 3)
```

FIGURE 19

EXAMPLE OF A TRACED FUNCTION

TABLE II

DIAGNOSTIC MESSAGES PRINTED BY THE NPS LISP SYSTEM

UNDEFINED FUNCTION - EVAL	The function is not defined on its property list or the A-list
UNDEFINED FUNCTION - APPLY	The function is not defined on its property list or the A-list
UNBOUND VARIABLE - EVAL	The atomic symbol being evaluated has no value binding on its property list or the A-list
CONDITIONAL UNSATISFIED - EVCON	All predicates in the conditional expression evaluate to F or NIL
ARGUMENTS ARE NOT LISTS OF EQUAL LENGTH - PAIR	PAIR requires that its two arguments be lists of equal length

(2) Since the first argument is not an atom, APPLY is called. The arguments of APPLY are the function (LAMBDA (A B) (CONS A B)), the list of arguments ((X Y Z) (1 2 3)), and the NIL A-list.

(3) APPLY recognizes LAMBDA, binds A to (X Y Z) and B to (1 2 3) on the A-list, and calls EVAL. The arguments of EVAL are the form (CONS A B) and the A-list ((A X Y Z) (B 1 2 3)).

(4) EVAL determines that CONS is a SUBR function and then evaluates A and B. The values of A and B are retrieved from the A-list.

(5) CONS is applied to its arguments, (X Y Z) and (1 2 3), and returns ((X Y Z) 1 2 3)).

(6) The computed value is returned to EVAL, APPLY, and EVALQUOTE in sequence.

C. NPS LISP FUNCTIONS

The NPS LISP system contains all of the elementary functions, logical connectives, defining and property list functions, table building and table reference functions, and list handling functions found in 7090 LISP. ADD1, SUB1, and NUMBERP are the only arithmetic functions implemented for the initial NPS LISP system. Descriptions of all implemented LISP functions are contained in Appendix A.

The development of the NPS LISP system spanned a period of five months. As the system was developed, considerable effort was expended to ensure an acceptably short response time. As a result, the LISP programmer can expect an almost immediate response when he inputs a pair of S-expressions for evaluation. The fact that the system was programmed in a higher level language, PL/I, provides major

advantages over a system written in machine code: (1) alterations to the system may be easily made; (2) the system may be implemented quite easily on another computer having a PL/I compiler; and (3) the system is self-documenting to some extent.

Recommendations for incorporating into NPS LISP some unimplemented features of 7090 LISP are discussed in the next, and final, section.

IV. RECOMMENDATIONS

Due to the magnitude of the task of implementing a LISP system, it was not possible, or even planned, to initially implement all of the features of 7090 LISP. However, the difficult design and lengthy programming tasks are complete, and the basic framework exists for expanding NPS LISP capabilities. Features of 7090 LISP recommended for incorporation into the NPS LISP system are discussed in the following paragraphs.

A. GARBAGE COLLECTION

Garbage collection is the process of reclaiming cells of inactive list structure when the free storage list becomes empty. It is one of the most significant features of 7090 LISP and desirable for any complete LISP system. In 7090 LISP, full word storage is also reclaimed during garbage collection.

When NPS LISP was designed, provisions were made in the cell structure for garbage collection of list structure. However, no garbage collection routines were written for the initial implementation. Bits one and 17 of each non-header cell are available for use as tags during the marking phase of garbage collection. The CAR of atom header cells contains -1; consequently, only the seventeenth bit of atom header cells can be used for marking.

The description of the garbage collection procedure utilized by 7090 LISP is in Ref. 2. In order to implement this scheme, all NPS

LISP functions written as recursive PL/I procedures will require some alteration. These functions must be rewritten to utilize a pushdown list for storing the partial results of computations. Otherwise, any list structures created by these functions are lost if garbage collection occurs during their execution. Functions similar to the 7090 LISP functions SAVE and UNSAVE could be written to maintain the pushdown list.

An alternative to rewriting the NPS LISP functions is available, however. A statement could be inserted in the LISP supervisor, LISPA, which prints a warning when the free storage list is dangerously close to becoming empty. When this message is seen by the LISP programmer, he then explicitly calls the garbage collection routine by typing a pair of S-expressions such as RECLAIM (). This solution is not as efficient or foolproof as conventional garbage collection methods; however, it provides a limited garbage collector with a minimum of programming effort.

B. ARITHMETIC FUNCTIONS

Three arithmetic functions are defined in NPS LISP -- NUMBERP, ADD1, and SUB1. The extensive arithmetic capabilities of 7090 LISP were not initially implemented for NPS LISP since they were not considered essential to the system operation. The implementation of most of the 7090 LISP arithmetic functions is not expected to be a major task due to the ease with which they may be written in PL/I.

The only significant problem which must be solved is the design of efficient methods of reading and storing floating point numbers.

The largest integer that may be stored in NPS LISP memory is $2^{31}-1$. Consequently, if two very large integers are multiplied together, a fixed point overflow may occur.

C. FUNCTIONS WITH FUNCTIONAL ARGUMENTS

The NPS LISP system does not contain the 7090 LISP functions MAP, MAPLIST, MAPCAR, or SEARCH. However, they may be defined by the LISP programmer through the use of the functions DEFINE and DEFLIST. It is recommended that these features be written as PL/I procedures and placed within the procedure NEVALQ since they require access to the current association list.

D. PROG FEATURE

The PROG feature allows the LISP programmer to write LISP programs in a convenient format similar to ALGOL. It was not implemented since any program that can be written using PROG can also be written in the standard LISP format. All LISP systems discussed in Section II, however, have the PROG feature. Thus, it would be desirable to implement this feature for NPS LISP.

The NPS LISP system at the current stage of development permits investigations in many areas applicable to list processing and symbol manipulation techniques. In addition, the system structure provides a good basis for future expanding the capabilities of NPS LISP.

APPENDIX A

FUNCTIONS IN THE NPS LISP SYSTEM

All LISP functions implemented for NPS LISP are described in this appendix. The effect, or result, of most NPS LISP functions is the same as the corresponding 7090 LISP function. Any significant differences are briefly explained. Each entry in this appendix consists of (1) the function name followed by its arguments; (2) an asterisk if the function is equivalent to the corresponding 7090 LISP function; (3) either SUBR or FSUBR; (4) an indication if the function is a predicate or pseudo-function; and (5) a brief description of the function. The functions are listed here in the same order as they are in Appendix A of Ref. 2 to allow easy comparisons to 7090 LISP functions. True and false are represented by T and F respectively.

ELEMENTARY FUNCTIONS

CAR(x) * SUBR

The value contained in bits 1-16 of the cell in FSTOR(x) is returned as the value of CAR(x).

CDR(x) * SUBR

The value represented by bits 17-32 of the cell in FSTOR(x) is returned as the value of CDR(x). CDR and CAR will always return a value even if applied to atomic arguments. The functions CAAR, CADR,...,CDDDR are also available as system functions.

CONS(x,y) SUBR

The value returned by CONS is a pointer to a cell obtained from the free storage list. The CAR of this cell contains x and the CDR of the cell contains y. CONS does not call a garbage collection routine when the free storage list is exhausted.

ATOM(x) * SUBR predicate

T is returned if x is an atom (i.e., pointer to an atom header); otherwise, F is returned by ATOM. Since the sign bit of an atom header cell is 1, ATOM tests for a negative value in FSTOR(x).

EQ(x,y) SUBR predicate

The value of EQ is T if x and y are the same atoms, and F otherwise. EQ is used for comparing numbers as well as literal atoms.

EQUAL(x,y) SUBR predicate

If x and y are the same S-expressions, the value of EQUAL is T; otherwise, the value is F. This function should not be used for comparing numbers.

LIST(x_1,x_2,...,x_n) * FSUBR

The value of LIST is a list of the arguments.

NULL(x) SUBR predicate

NULL returns T if x is NIL; otherwise it returns F. In contrast to 7090 LISP, NIL is not equivalent to zero.

RPLACA(x,y) * SUBR

The CAR of x is replaced by y; the value x is returned.

RPLACD(x,y) * SUBR

The CDR of x is replaced by y; the value x is returned. RPLACA and RPLACD should be used with caution since they permanently alter list structure.

LOGICAL CONNECTIVES

AND(x_1,x_2,...x_n) FSUBR predicate

The arguments of AND are evaluated in sequence until one of them evaluates to F or NIL. The value of AND is F in this case or T if all of the arguments evaluate to some value other than F or NIL.

OR(x_1,x_2,...,x_n) FSUBR predicate

The arguments of OR are evaluated in order until one of them evaluates to some value other than F or NIL. The value of OR is T in this case or F if all arguments evaluate to F or NIL.

NOT(x) SUBR predicate

The value of NOT is T if x is F or NIL; otherwise, the value of NOT is F.

DEFINING AND PROPERTY LIST FUNCTIONS

DEFINE(x) * SUBR pseudo-function

The argument of DEFINE is a list of pairs; each pair consists of a function name and an S-expression definition of the function. DEFINE puts an EXPR indicator and a pointer to the corresponding function definition on the property list of each name. A list of the function names is returned.

DEFLIST(x,i) * SUBR pseudo-function

This function works like define except that it puts the indicator i on the property lists instead of EXPR. DEFLIST can be used to put any property on a property list. If either DEFINE or DEFLIST is used more than once on the same property list, the most recent property under any particular indicator replaces the previous one. For example, if a function is re-defined several times using DEFINE, only the most recent definition is on the function's property list.

ATTRIB(x,y) * SUBR pseudo-function

The last element of x is changed by ATTRIB to the value y; the value returned is y. ATTRIB can be used to attach something onto the end of a property list.

PROP(x,y,u) SUBR

The function PROP searches the list x for an element that is EQ to y. If such an element is found, the value of PROP is the remainder of the list beginning immediately after y. Otherwise, the value of PROP is u. The argument u is any S-expression.

GET(x,y) * SUBR

GET searches the list x for an element that is EQ to y. The CAR of the remainder of the list immediately following y is returned if the search is successful. Otherwise, the value of GET is NIL.

CSET(x,v) * SUBR pseudo-function

This pseudo-function establishes a zero-level binding of the atom x to the value v; it puts the indicator APVAL and a value cell whose CAR points to v on the property list of x. The value of CSET is x.

CSETQ(x,v) * SUBR pseudo-function

CSETQ is similar to CSET except that it quotes the first argument, x, instead of evaluating it. CSETQ cannot be input at the top level of EVALQUOTE. That is, the following pair of S-expressions is illegal: CSETQ (A (1 2 3)).

REMPROP(x,i) * SUBR pseudo-function

This pseudo-function deletes all occurrences of the indicator i and its related property from the list x. The value of REMPROP is NIL.

FLAG(l,i) * SUBR pseudo-function

The argument l is a list of atoms. FLAG puts the flag i on the property list of each atom in list l. The value of FLAG is NIL.

REMFLAG(l,i) * SUBR pseudo-function

REMFLAG has the opposite effect of FLAG. It removes all occurrences of the flag i from the property list of each atom in the list l and returns NIL.

TABLE BUILDING AND TABLE REFERENCE FUNCTIONS

PAIR(x,y) * SUBR

The function PAIR returns a list of dotted pairs of corresponding elements of the lists x and y.

PAIRLIS(x,y,a) * SUBR

PAIRLIS acts like PAIR with the additional property that it adds the list of pairs to the front of list a. The value of PAIRLIS is the modified list a.

SASSOC(x,y,u) SUBR

The function SASSOC searches the list of dotted pairs, y, for a pair whose first element is x. If such a pair is found, the value of SASSOC is this pair. Otherwise, the S-expression u is returned.

ASSOC(x,y) SUBR

ASSOC, like SASSOC, returns the pair on the list y whose first term is EQ to x. If no such pair is found, the value of PAIR is NIL.

SUBST(x,y,z) * SUBR

SUBST returns the new S-expression formed by substituting x for all occurrences of y in the S-expression z. Note that SUBST does not change the S-expression z.

SUB2(x,y) SUBR

SUB2 is similar to ASSOC. It searches the list of pairs x for a pair whose first term is EQ to y. SUB2 returns the second term of the pair if the search is successful; otherwise, it returns y.

SUBLIS(x,y) * SUBR

The argument x is a list of dotted pairs of the form $((u_1.v_1)(u_2.v_2)$ $...(u_n.v_n))$ and y is an S-expression. SUBLIS returns a new S-expression formed by replacing all occurrences of any u_i's in y by their corresponding v_i's. The S-expression y is not altered.

LIST MANIPULATING FUNCTIONS

APPEND(x,y) * SUBR

The function APPEND attaches the list y to the end of a copy of the list x and returns the resultant list.

NCONC(x,y) * SUBR

NCONC concatenates the lists x and y without copying the list x; it returns the combined list. This function utilizes the function RPLACD; thus, list structure is permanently altered.

COPY(x) * SUBR

The function COPY returns a copy of the list x.

REVERSE(x) * SUBR

This function reverses the top level of the list x and returns the reversed list.

MEMBER(x,y) * SUBR predicate

If the S-expression x is a member of the list y, MEMBER returns T; otherwise it returns F.

LENGTH(x) * SUBR

The length of the top level of the list x is returned by LENGTH.

LAST(x) SUBR

The last element of the list x is returned as the value of LAST.

EFFACE(x,y) * SUBR

EFFACE alters the list y by removing from it all occurrences of the item x.

ARITHMETIC FUNCTIONS

NUMBERP(x) * SUBR predicate

The predicate NUMBERP returns T if X is a numerical atom and F otherwise. It utilizes the fact that number values are stored in full word storage (i.e., the region of FSTOR above the pointer BNUM).

ADD1(x) * SUBR

The argument x must be a numerical atom; if ADD1 (or SUB1) is applied to a literal atom, the property list of the literal atom will be lost. ADD1 adds one (1) to the value of x and returns this new value.

SUB1(x) * SUBR

The function SUB1 operates similarly to ADD1 except that the value of (SUB1 X) is X-1.

INPUT AND OUTPUT FUNCTIONS

READ() SUBR pseudo-function

When READ is executed, the next S-expression in the input buffer is read into the system. READ puts atoms into the OBLIST and creates atom property lists. The value of READ is the S-expression read.

PRINT(x) * SUBR pseudo-function

The execution of PRINT causes the S-expression x to be printed; the value of PRINT is x.

PRIN1(x) * SUBR pseudo-function

The pseudo-function PRIN1 puts the atom x on the print buffer; the print buffer is not printed unless it is full.

TERPRI() * SUBR pseudo-function

TERPRI prints the current contents of the print buffer. TERPRI also clears the print buffer after printing it and returns NIL.

FUNCTIONS FOR DEBUGGING

TRACE(x) * SUBR pseudo-function

TRACE puts a flag on the property list of each function in the list x which causes trace information to be printed when the function is entered and exited. The argument x must be a list of function names. The value of TRACE is NIL.

UNTRACE(x) * SUBR pseudo-function

UNTRACE removes the trace flag from the property lists of all function names in the list x. The value of UNTRACE is NIL.

INTERPRETER FUNCTIONS

APPLY(fn,args,a) * SUBR

The arguments of APPLY are a function, a list of arguments for the function, and the current association list. APPLY applies a function to its arguments and returns the resultant value.

EVAL(form,a) * SUBR

The arguments of EVAL are a form to be evaluated and the current association list. The value returned by EVAL is the value of the form.

EVLIS(l,a) * SUBR

EVLIS evaluates a list of forms; the list is usually a list of arguments for a function which are evaluated before the function is applied. The arguments of EVLIS are a list of forms for evaluation and the current association list. EVLIS returns a list of values of the forms.

EVCON(c,a) SUBR

The arguments of EVCON are a list of pairs of S-expressions; for each pair, the first element is a predicate while the second element is an S-expression that is evaluated if the value of the predicate expression is not F or NIL. The value of EVCON is the value of the second S-expression in this case. At least one of the predicates must evaluate to a value other than F or NIL; otherwise, an error results.

APPENDIX B

USING THE NPS LISP SYSTEM

The NPS LISP system is a PL/I program that runs under control of the Cambridge Monitor System (CP/CMS). NPS LISP consists of six external PL/I procedures which reside on a private user disk file under the file names LISPA, LISPB, LISPC, LISPD, LISPE, and LISPP. LISPA is the NPS LISP supervisor; it reads two S-expressions, calls EVALQUOTE, and prints the value of EVALQUOTE. LISPB contains most of the functions implemented for NPS LISP. LISPC contains the read routines. LISPD is the interpreter, NEVALQ. LISPE is the initialization routine, INITIAL, which puts all system functions in memory. LISPP contains the elementary LISP functions.

A user of NPS LISP must log on to CP in the usual manner. He then must enter an 'IPL CMS' command and, when CMS is loaded, type 'LISP'. (Single quotes are used here to specify what is actually typed by the user). This causes system initialization. The system is ready to accept input when the terminal types 'CALL EVALQUOTE, ARGS:', skips to the next line, and types '_'.

EVALQUOTE is not called until two complete S-expressions have been read. The character '_' is typed if any input line is not complete. When EVALQUOTE has completed execution, the terminal types 'VALUE IS:' followed by the value of EVALQUOTE. A listing of a brief session with the NPS LISP system follows:

```
ipl cms
CMS..VERSION 11.0 - 06/15/69

lisp
EXECUTION BEGINS...

NPS LISP INITIALIZING

CALL EVALQUOTE, ARGS:
_(lambda (x) x) ((this is an example of nps lisp))

VALUE IS:
(THIS IS AN EXAMPLE OF NPS LISP)


CALL EVALQUOTE, ARGS:
_define ((

_( intersection (lambda (x y) (cond ((null x) nil)

_      ((member (car x) y) (cond@s (car x) (intersection

_      (cdr x) y))) (t (intersection (cdr x) y)) )))

_(atomlist (lambda (l) (cond ((null l) t)

_      ((atom (car l)) (atomlist (cdr l))) (t f)))) ))

VALUE IS:
(INTERSECTION ATOMLIST)


CALL EVALQUOTE, ARGS:
_intersection ((a b d e j t r s)(t e s a 1 5 j))

VALUE IS:
(A E J T S)


CALL EVALQUOTE, ARGS:
_atomlist ((a b c d e))

VALUE IS:
T


CALL EVALQUOTE, ARGS:
_atomlist ((a b (c) d e))
```

```
VALUE IS:
NIL


CALL EVALQUOTE, ARGS:
_cset (lista (a b c d e))

VALUE IS:
LISTA


CALL EVALQUOTE, ARGS:
_(lambda (x) (cons x (cddr lista))) ((1 2 3))

VALUE IS:
((1 2 3) C D E)


CALL EVALQUOTE, ARGS:
_reverse ((a b c d e))

VALUE IS:
(E D C B A)


CALL EVALQUOTE, ARGS:
_pair ((a b c d)(1 2 3 4))

VALUE IS:
((A . 1) (B . 2) (C . 3) (D . 4))


CALL EVALQUOTE, ARGS:
_add1 (789)

VALUE IS:
790


CALL EVALQUOTE, ARGS:
_subst (one 1 (1 2 (4 1)((7 3 1 3) 1 6)))))))))

VALUE IS:
(ONE 2 (4 ONE) ((7 3 ONE 3) ONE 6))


CALL EVALQUOTE, ARGS:
_end lisp
EXIT LISP SYSTEM
R; T=20.09/27.54 15.14.33
```

APPENDIX C

LISTING OF NPS LISP SYSTEM

```
LISPA: PROC OPTIONS(MAIN);
       /* THIS IS THE LISP SUPERVISOR ROUTINE */
       DCL NREAD ENTRY EXTERNAL,
       (S1,S2,S3) FIXED BIN;
       CALL INITIAL;
  LOOP: S1=NREAD; IF S1<0 THEN GO TO TERM;
             IF S1=127 THEN GOTO LOOP;
       S2=NREAD; IF S2<0 THEN GO TO TERM;
             IF S2=127 THEN GOTO LOOP;
       S3=NEVALQ(S1,S2);
       S3=NPRINT(S3);
       DISPLAY('   '); DISPLAY('   ');
       GO TO LOOP;
  TERM: DISPLAY('EXIT LISP SYSTEM');
       RETURN;
       END LISPA;
```

```
LISPP: PROC;

        /* LISP PRIMITIVE (ELEMENTARY) FUNCTIONS */
    DECLARE
    FSTOR(0:16000) FIXED BIN(31) STATIC EXT,
    (BFREE,TRACE,TRCCNS) FIXED BIN STATIC EXTERNAL,
    (GB32 BIT (32), GB16 BIT(16)) STATIC,
    (NC,FREE ,GT1) BIN FIXED STATIC EXTERNAL,
     (T,F,NIL,BNUM,MFSTOR) FIXED BIN STATIC EXT;

NCELL: ENTRY;
      /* GETS THE NEXT AVAILABLE CELL IN FREE STORAGE */
      NC=FREE; FREE=FSTOR(FREE);
      RETURN(NC);

 NCAR: ENTRY(CA);
      DCL (CA) FIXED BIN;
      GB16=SUBSTR(UNSPEC(FSTOR(CA)),1,16);
      GT1=GB16;
      RETURN(GT1);

 NCDR: ENTRY(CD);
      DCL (CD) FIXED BIN;
      GB16=SUBSTR(UNSPEC(FSTOR(CD)),17,16);
      GT1=GB16;
      RETURN(GT1);

NCONS: ENTRY(CNA,CNB);
      DCL (CNA,CNB) FIXED BIN;
      GB32=UNSPEC(CNB);
      GB32=GB32 | SUBSTR(UNSPEC(CNA),17,16);
      GT1=NCELL;
      FSTOR(GT1)=GB32;
      RETURN(GT1);

 NATOM: ENTRY(JAT);
      DCL JAT;
        IF JAT>MFSTOR THEN RETURN(F);
        IF JAT=NIL THEN RETURN(T);
        IF FSTOR(JAT)<0 THEN RETURN(T);
        RETURN(F);

 NEQ: ENTRY(JEQ,KEQ);
      DCL (JEQ,KEQ) FIXED BIN;
        IF NATOM(JEQ)=T THEN
             IF NATOM(KEQ)=T THEN
```

```
                    IF JEQ=KEQ THEN RETURN(T);
                    IF NCDR(JEQ)>BNUM THEN
                       IF NCDR(KEQ)>BNUM THEN
                          IF FSTOR(NCDR(JEQ))=FSTOR(NCDR(KEQ))
                          THEN RETURN(T);
          RETURN(F);

NEQUAL: ENTRY(JEQL,KEQL) RECURSIVE;
          /*  RETURNS T IF J AND K ARE THE SAME S-EXPRESSIONS
              AND F OTHERWISE  */
              DCL (JEQL,KEQL);
              IF NATOM(JEQL)=T & NATOM(KEQL)=T THEN
                 RETURN(NEQ(JEQL,KEQL));
              IF NATOM(JEQL)=T | NATOM(KEQL)=T THEN
                 RETURN(F);
              IF NEQUAL(NCAR(JEQL),NCAR(KEQL))=T THEN
                 RETURN(NEQUAL(NCDR(JEQL),NCDR(KEQL)));
              RETURN(F);

NULL: ENTRY(NLL);
          /*  RETURNS T IF NLL IS THE NULL LIST, F OTHERWISE  */
              DCL NLL;
              IF NLL=NIL THEN RETURN(T);
              RETURN(F);

NRPLACA: ENTRY(JCA,KCA);
          /*  REPLACES CAR(JCA) WITH KCA  -  VALUE IS JCA  */
              DCL (JCA,KCA);
              GB32='00000000000000001111111111111111'B;
              GB32=GB32 & UNSPEC(FSTOR(JCA));
              GB32=GB32 | SUBSTR(UNSPEC(KCA),17,16);
              FSTOR(JCA)=GB32;
              RETURN(JCA);

NRPLACD: ENTRY(JCD,KCD);
          /*  REPLACES CDR(JCD) WITH KCD  -  VALUE IS JCD  */
              DCL (JCD,KCD);
              IF JCD=NIL THEN RETURN(NIL);
              GB32=SUBSTR(UNSPEC(FSTOR(JCD)),1,16) | UNSPEC(KCD);
              FSTOR(JCD)=GB32;
              RETURN(JCD);

   END LISPP;
```

```
LISPB: PROC;

        /* LISP SUBR FUNCTIONS.  ENTRY POINT NAMES CORRESPOND TO LISP
           FUNCTION NAMES WITH AN 'N' PRECEEDING THEM. */
    DECLARE
    FSTOR(0:16000) FIXED BIN(31) STATIC EXT,
    (PNAME,EXPR) FIXED BIN STATIC EXT,
    (T,F,NIL,MFSTOR,BNUM,MONE INITIAL(-1),FLAG) FIXED BIN STATIC EXT;

NCAAR: ENTRY(KAA); DCL KAA; RETURN(NCAR(NCAR(KAA)));

NCADR: ENTRY(KAD); DCL KAD; RETURN(NCAR(NCDR(KAD)));

NCDAR: ENTRY(KDA); DCL KDA; RETURN(NCDR(NCAR(KDA)));

NCDDR: ENTRY(KDD); DCL KDD; RETURN(NCDR(NCDR(KDD)));

NCAAAR: ENTRY(KAAA); DCL KAAA; RETURN(NCAR(NCAAR(KAAA)));

NCAADR: ENTRY(KAAD); DCL KAAD; RETURN(NCAR(NCADR(KAAD)));

NCADAR: ENTRY(KADA); DCL  KADA; RETURN(NCAR(NCDAR(KADA)));

NCDAAR: ENTRY(KDAA); DCL KDAA; RETURN(NCDR(NCAAR(KDAA)));

NCADDR: ENTRY(KADD); DCL KADD; RETURN(NCAR(NCDDR(KADD)));

NCDADR: ENTRY(KDAD); DCL KDAD; RETURN(NCDR(NCADR(KDAD)));

NCDDAR: ENTRY(KDDA); DCL KDDA; RETURN(NCDR(NCDAR(KDDA)));

NCDDDR: ENTRY(KDDD); DCL KDDD; RETURN(NCDR(NCDDR(KDDD)));

NOT: ENTRY(NOTARG);
        /* RETURNS F IF NOTARG IS TRUE, OTHERWISE RETURNS T */
             DCL NOTARG;
             IF NOTARG=F | NOTARG=NIL THEN RETURN(T);
             RETURN(F);

NDEFINE: ENTRY(LDEF) RECURSIVE;
        /*  LDEF IS A LIST OF PAIRS OF NAMES AND LAMBDA EXPRESSIONS.
            NDEFINE PUTS AN 'EXPR' INDICATOR AND THE LAMBDA EXPRESSION
            ON THE PROPERTY LIST FOR EACH NAME.  THE VALUE OF NDEFINE
            IS A LIST OF NAMES DEFINED IN THIS WAY  */

             DCL (LDEF,KDEFT,LDEFT);
```

```
                IF LDEF=NIL THEN RETURN(NIL);
                LDEFT=NPROP(NCAAR(LDEF),PNAME,NIL);
                KDEFT=NREMPRP(NCAAR(LDEF),EXPR);
                KDEFT=NCDR(LDEFT);
                KDEFT=NCONS(NCAR(NCDAR(LDEF)),KDEFT);
                KDEFT=NCONS(EXPR,KDEFT);
                KDEFT=NRPLACD(LDEFT,KDEFT);
                RETURN(NCONS(NCAAR(LDEF),NDEFINE(NCDR(LDEF))));

NDEFLST: ENTRY(NDEF,NIND) RECURSIVE;
        /*  THIS FUNCTIONS WORKS JUST LIKE NDEFINE EXCEPT THE INDICATOR
            'NIND' SUPPLIED AS AN ARGUMENT IS PUT ON THE PROPERTY LIST
           RATHER THAN 'EXPR'.  VALUE IS A LIST OF NAMES DEFINED  */

                DCL (NDEF,NIND,MDEFT,NDEFT);
                IF NDEF=NIL THEN RETURN(NIL);
                NDEFT=NPROP(NCAAR(NDEF),PNAME,NIL);
                MDEFT=NREMPRP(NCAAR(NDEF),NIND);
                MDEFT=NCDR(NDEFT);
                MDEFT=NCONS(NCAR(NCDAR(NDEF)),MDEFT);
                MDEFT=NCONS(NIND,MDEFT);
                MDEFT=NRPLACD(NDEFT,MDEFT);
                RETURN(NCONS(NCAAR(NDEF),NDEFLST(NCDR(NDEF),NIND)));

NATTRIB: ENTRY(JATR,KATR);
        /*  ATTACHES LIST K ONTO END OF LIST J AND RETURNS K  */
                DCL (JATR,KATR,N);
                N=NCONC(JATR,KATR);
                RETURN(KATR);

 NPROP: ENTRY (JPR,KPR,IPFN);      /* 11-12-96 */
        /* RETURNS CDR OF CELL WHOSE CAR IS EQ TO KPR IF ONE EXISTS,
                RETURNS THE FUNCTION IPFN OTHERWISE */
                DCL (JPR,JPR1,KPR,IPFN) FIXED BIN;
                IF NATOM(JPR)=T THEN JPR1=NCDR(JPR); ELSE JPR1=JPR;
        PROP:IF JPR1=NIL THEN RETURN(IPFN);
                IF NEQ(NCAR(JPR1),KPR)=T THEN RETURN(NCDR(JPR1));
                JPR1=NCDR(JPR1);
                GO TO PROP;

NGET: ENTRY(JGET,KGET);     /* 11-12-69 */
        /* RETURNS CADR OF CELL WHOSE CAR IS EQUAL TO KGET IF ONE
                EXISTS, RETURNS NIL OTHERWISE */
                DCL (JGET,JGET1,KGET,KGET1) FIXED BIN;
                IF NATOM(JGET)=T THEN JGET1=NCDR(JGET);
                                 ELSE JGET1=JGET;
```

```
        GETA: IF JGET1=NIL THEN RETURN(NIL);
              IF NEQ(NCAR(JGET1),KGET)=T THEN RETURN(NCADR(JGET1));
              JGET1=NCDR(JGET1);
              GO TO GETA;

NREMPRP: ENTRY(NRM,NRMP) ;
         /*  DELETES ALL OCCURRENCES OF INDICATOR AND RELATED
             PROPERTY FROM PROP. LIST OF ATOM NRM */
             DCL (NRM,NRMP,NRMP1,NRM2);
             NRMP1=NRM;
             DO WHILE (NCDR(NRMP1)¬=NIL);
                  IF NEQ(NCADR(NRMP1),NRMP)=T THEN
                       NRM2=NRPLACD(NRMP1,NCDDR(NRMP1));
                  ELSE NRMP1=NCDR(NRMP1);
             END;
             RETURN(NIL);

NFLAG: ENTRY(FLST,FID);
         /* PUTS THE FLAG INDICATOR 'FID' ON THE PROPERTY LIST
            OF EACH ATOMIC SYMBOL ON LIST 'FLST' IMMEDIATELY
            FOLLOWING THE ATOM HEADER  */
             DCL (FLST,FID,FLST1,FLST2) FIXED BIN;
             FLST1=FLST;
          NFLG:IF FLST1=NIL THEN RETURN(NIL);
             FLST2=NRPLACD(NCAR(FLST1),NCONS(FID,NCDAR(FLST1)));
             FLST1=NCDR(FLST1);
             GO TO NFLG;

NREMFLG: ENTRY(RMFG,RID);
         /* REMOVES ALL OCCURRENCES OF FLAG 'RID' FROM THE PROPERTY
            LIST OF ALL ATOMIC SYMBOLS ON LIST 'RMFG  */
             DCL (RMFG,RID,RMFG1,RMFG2,NFG) FIXED BIN;
             RMFG1=RMFG;
             DO WHILE (RMFG1¬=NIL);
                  RMFG2=NCAR(RMFG1);
                  DO WHILE (NCDR(RMFG2)¬=NIL);
                       IF NEQ(NCADR(RMFG2),RID)=T THEN
                            NFG=NRPLACD(RMFG2,NCDDR(RMFG2));
                  ELSE RMFG2=NCDR(RMFG2);
                  END;
                  RMFG1=NCDR(RMFG1);
             END;
            RETURN(NIL);

NPAIR: ENTRY(JPAR,KPAR) ;
         /*  RETURNS LIST OF DOTTED PAIRS OF CORRESPONDING ELEMENTS OF
```

```
                LISTS JPAR AND KPAR WHICH MUST BE OF EQUAL LENGTH  */
                DCL (JPAR,KPAR,LPAR,MPAR,NPAR);
                LPAR=NIL;
                MPAR=JPAR; NPAR=KPAR;
        PAIRA:  IF MPAR=NIL | NPAR=NIL THEN
                DO;  IF MPAR=NIL THEN
                          IF NPAR=NIL THEN RETURN(NREVERS(LPAR));
                     RETURN(NIL);
                END;
                LPAR=NCONS(NCONS(NCAR(MPAR),NCAR(NPAR)),LPAR);
                MPAR=NCDR(MPAR); NPAR=NCDR(NPAR);
                GO TO PAIRA;

NPAIRLS: ENTRY(JPAIR,KPAIR,LPAIR) RECURSIVE;
        /*  CREATES A LIST OF PAIRS OF CORRESPONDING ELEMENTS OF
           LISTS JPAIR AND KPAIR AND PUTS ON FRONT OF LIST LPAIR*/
                DCL (JPAIR,KPAIR,LPAIR,MPAIR,NPAIR);
                IF JPAIR=NIL THEN RETURN(LPAIR);
                MPAIR=NCONS(NCAR(JPAIR),NCAR(KPAIR));
                NPAIR=NPAIRLS(NCDR(JPAIR),NCDR(KPAIR),LPAIR);
                RETURN(NCONS(MPAIR,NPAIR));

NSASSOC: ENTRY(JSAS,KSAS,LFN);
        /*  RETURNS DOTTED PAIR FROM LIST KSAS WHOSE CAR IS EQ
                TO JSAS, IF NO SUCH PAIR EXISTS, RETURNS THE
                FUNCTION LFN  */
                DCL (JSAS,KSAS,LFN,MSAS);
                MSAS=KSAS;
        SAS:    IF MSAS=NIL THEN RETURN(LFN);
                IF NEQ(NCAAR(MSAS),JSAS)=T THEN RETURN(NCAR(MSAS));
                MSAS=NCDR(MSAS);
                GO TO SAS;

NASSOC: ENTRY(JAS,LAS);
        /*  RETURNS POINTER TO PAIR ON LIST LAS (A-LIST) WHOSE
                FIRST TERM = JAS */
                DCL (JAS,LAS);
        NASS:IF NEQUAL(NCAAR(LAS),JAS)=T THEN RETURN(NCAR(LAS));
                LAS=NCDR(LAS);
                IF LAS=NIL THEN RETURN(NIL);
                GO TO NASS;

NSUBST: ENTRY(JST,KST,LSBST) RECURSIVE;
        /*  RETURNS LIST FORMED BY SUBSTITUTING JST FOR EVERY
                OCCURRENCE OF KST IN LSBST  */
                DCL (JST,KST,LSBST);
```

```
            IF NEQUAL(KST,LSBST)=T THEN RETURN(JST);
            IF NATOM(LSBST)=T THEN RETURN(LSBST);
            RETURN(NCONS(NSUBST(JST,KST,NCAR(LSBST)),
                NSUBST(JST,KST,NCDR(LSBST))));

NSUB2: ENTRY(LSUB,KSUB) RECURSIVE;
       /*  RETURNS SECOND TERM OF PAIR ON LIST LSUB HAVING FIRST
            TERM EQUAL TO KSUB  */
            DCL (LSUB,KSUB);
            IF LSUB=NIL THEN RETURN(KSUB);
            IF NEQ(NCAAR(LSUB),KSUB)=T THEN RETURN(NCDAR(LSUB));
            RETURN(NSUB2(NCDR(LSUB),KSUB));

NSUBLIS: ENTRY(LSBL,KSBL) RECURSIVE;
       /*  RETURNS AN S-EXPRESSION  IN WHICH ALL VARIABLES IN
            S-EXP. KSBL HAVE BEEN REPLACED BY THE VALUES TO
            WHICH THEY ARE CURRENTLY BOUND ON THE A-LIST (LSBL) */
            DCL (LSBL,KSBL);
            IF NATOM(KSBL)=T THEN RETURN(NSUB2(LSBL,KSBL));
            RETURN(NCONS(NSUBLIS(LSBL,NCAR(KSBL)),
                    NSUBLIS(LSBL,NCDR(KSBL))));

NAPPEND: ENTRY(JAPP,KAPP) RECURSIVE;
       /*  APPENDS LIST KAPP TO END OF LIST JAPP  -  VALUE RETURNED
            IS NEW LIST  */
            DCL (JAPP,KAPP);
            IF JAPP=NIL THEN RETURN(KAPP);
            RETURN(NCONS(NCAR(JAPP),NAPPEND(NCDR(JAPP),KAPP)));

NCONC: ENTRY(JNC,KNC) ;
       /*  RETURNS LIST JNC WITH LIST KNC ADDED ON TO THE END   */
            DCL (JNC,KNC,MNC,NCONA);
            IF JNC=NIL THEN RETURN(KNC);
            MNC=JNC;
       NCON:IF NCDR(MNC)=NIL THEN
                 DO; NCONA=NRPLACD(MNC,KNC); RETURN(JNC); END;
            MNC=NCDR(MNC);
            GO TO NCON;

NCOPY:ENTRY(JCOP) RECURSIVE;
       /*  RETURNS A COPY OF LIST JCOP  */
            DCL (JCOP,MCOP,NCOP);
            IF JCOP=NIL THEN RETURN(NIL);
            IF NATOM(JCOP)=T THEN RETURN(JCOP);
            MCOP=NCOPY(NCAR(JCOP));
            NCOP=NCOPY(NCDR(JCOP));
```

```
                RETURN(NCONS(MCOP,NCOP));

NREVERS: ENTRY(LSTRU);
        /* RETURNS THE REVERSE OF THE TOP LEVEL OF LIST LSTRU */
                DCL (LSTRU,LSTRV);
                LSTRV=NIL;
        AREV:IF LSTRU=NIL THEN RETURN(LSTRV);
                LSTRV=NCONS(NCAR(LSTRU),LSTRV);
                LSTRU=NCDR(LSTRU); GO TO AREV;

MEMBER: ENTRY(JMEM,KMEM) RECURSIVE;
        /*  RETURNS T IF LIST JMEM IS A MEMBER IF LIST KMEM,
                F OTHERWISE  */
                DCL (JMEM,KMEM);
                IF KMEM=NIL THEN RETURN(F);
                IF NEQUAL(JMEM,NCAR(KMEM))=T THEN RETURN(T);
                RETURN(MEMBER(JMEM,NCDR(KMEM)));

 LENGTH: ENTRY(LGTH) ;
        /*  RETURNS LENGTH OF TOP LEVEL OF LIST LGTH  */
                DCL (LGTH,KLG,NLG);
                KLG=LGTH;
                NLG=0;
                DO WHILE(KLG¬=NIL);  NLG=NLG+1; KLG=NCDR(KLG); END;
                FSTOR(BNUM)=NLG;
                NLG=NCONS(MONE,BNUM);
                BNUM=BNUM-1;
                RETURN(NLG);

LAST: ENTRY(JLST) ;
        /*  RETURNS THE LAST ITEM ON LIST JLST  */
                DCL (JLST,KLST);
                KLST=JLST;  IF JLST=NIL THEN RETURN(NIL);
                DO WHILE(NCDR(KLST)¬=NIL); KLST=NCDR(KLST); END;
                RETURN(KLST);

NEFFACE: ENTRY(JEF,KEF)  ;
        /* DELETES ITEM JEF FROM THE TOP LEVEL OF LIST KEF AND
                RETURNS ALTERED LIST KEF  */
                DCL (JEF,KEF,KEF1,EF1,EF2) FIXED BIN;
                KEF1=KEF;
                DO WHILE (NEQUAL(JEF,NCAR(KEF1))=T);
                     KEF1=NCDR(KEF1); END;
                EF2=KEF1; EF1=NCDR(KEF1);
                DO WHILE (EF1¬=NIL);
                     IF NEQUAL(NCAR(EF1),JEF)=T THEN
```

```
                    DO;  EF1=NCDR(EF1);
                         EF2=NRPLACD(EF2,EF1);
                    END;
                    ELSE DO; EF2=EF1; EF1=NCDR(EF2); END;
               END;
               RETURN(KEF1);

NUMBERP: ENTRY(NUMP) ;
         /*  RETURNS T IF NUMP IS AN INTEGER NUMBER  -  F OTHERWISE  */
               DCL NUMP;
               IF NCDR(NUMP)>BNUM THEN RETURN(T);
               RETURN(F);

NADD1: ENTRY(NAD1);
         /* ADDS ONE TO THE VALUE REPRESENTED BY NAD1 */
               DCL (NAD1,NAD2);
               FSTOR(BNUM)=FSTOR(NCDR(NAD1))+1;
               NAD2=NCONS(MONE,BNUM);
               BNUM=BNUM-1;
               RETURN(NAD2);

NSUB1: ENTRY(NSB1);
         /* SUBTRACTS ONE FROM THE VALUE REPRESENTED BY NSB1 */
               DCL (NSB1,NSB2);
               FSTOR(BNUM)=FSTOR(NCDR(NSB1))-1;
               NSB2=NCONS(MONE,BNUM);
               BNUM=BNUM-1;
               RETURN(NSB2);

NTRACE: ENTRY(TRLST);
               /* PUTS TRACE INDICATOR ('FLAG') ON THE PROPERTY LIST
                    OF EACH ATOMIC SYMBOL IN LIST TRLST */
               DCL TRLST FIXED BIN;
               RETURN(NFLAG(TRLST,FLAG));

NUNTRCE: ENTRY(UTRL);
         /* REMOVES TRACE INDICATORS PLACED BY FUNCTION TRACE */
               DCL UTRL FIXED BIN;
               RETURN(NREMFLG(UTRL,FLAG));

END LISPB;
```

```
LISPC: PROC;

        /* READ ROUTINES - LEXICAL AND SYNTAX ANALYSIS */

 NREAD: ENTRY BIN FIXED;

 /* THIS PROCEDURE IS CALLED ONCE PER S-EXPRESSION. IT RETURNS THE
    POINTER TO THE S-EXPRESSION TREE IT HAS BUILT. IT RETURNS A
    "-1" ON AN "END LISP" INPUT. */

        DECLARE
        ACCUM CHAR(30) EXTERNAL,
        BP BIN FIXED EXTERNAL,
        BUFFER CHAR(72) EXTERNAL,
        BUFFNO BIN FIXED EXTERNAL,
        CADD BIN FIXED EXTERNAL,
        COUNT BIN FIXED EXTERNAL,
        CTR BIN FIXED EXTERNAL,
        DOTTAG BIT(1) STATIC ALIGNED,
        LDIG FIXED BIN(31) STATIC,
        JJ BIN FIXED EXTERNAL,
        LINE BIN FIXED EXTERNAL,
        NIL BIN FIXED EXTERNAL,
        N127 FIXED BIN STATIC,
        MODE BIN FIXED EXTERNAL,
        MODETAG BIT(1) EXTERNAL,
        MONE BIN FIXED INITIAL(-1) EXTERNAL,
        BNUM BIN FIXED EXTERNAL,
        NEVAL3 ENTRY,
        NUNSTK ENTRY,
        PLEV BIN FIXED EXTERNAL,
        POSIT BIN FIXED(31) EXTERNAL,
        RESULT BIN FIXED EXTERNAL,
        SEXP BIN FIXED EXTERNAL,
        STAC(200) BIN FIXED(31) EXTERNAL,
        STKTAB BIT(1) EXTERNAL,
        SWTCH(0:3) LABEL(STP,ATM,DIG,SPEC)
                   INITIAL(STP,ATM,DIG,SPEC),
        T BIN FIXED EXTERNAL,
        TCOUNT BIN FIXED EXTERNAL;

        N127=127;
        DOTTAG='0'B;
        IF ¬STKTAB THEN
             DO; BP=0; JJ=-1; BUFFNO=BUFFNO+1;
             END;
```

```
      ELSE DO; IF ¬MODETAG THEN
           DO; TCOUNT=0; BP=72;
           END;
      CTR=200; SEXP,PLEV=0; JJ=1;
      END;
CONT: CALL SCAN;
      GOTO SWTCH(RESULT);
 STP: RETURN(MONE);     /* "END LISP" WAS INPUT */
 ATM: POSIT=MOD(UNSPEC(SUBSTR(ACCUM,1,3))+COUNT+   /* HASH CODE */
           16777214,N127)+1;
      IF ¬STKTAB THEN
           DO; IF BUFFNO=2 | BUFFNO=3 THEN
                DO; L=LINE; CALL INIT1(L);
                GOTO IN;
                END;
           ELSE GOTO IN;
           END;
      IF JJ<0 THEN
  IN: DO; CALL ENTER(POSIT); JJ=1; IF STKTAB THEN
           DO; L=NEVAL2; IF L<0 THEN
                DO; RETURN(SEXP); END;
           END;
      ELSE DO; L=LINE; LINE=LINE+1;
      IF BUFFNO>4 THEN
           IF BUFFNO=13 THEN CALL INITL3(L);
      ELSE CALL INITL2(L);
      ELSE CALL INITL(L);
      END;
      GOTO CONT;
      END;
      JJ=LOOKUP(POSIT);
      IF JJ<0 THEN GOTO IN;
 ELSE DO; IF STKTAB THEN
           DO; L=NEVAL2; IF L<0 THEN
                DO; RETURN(SEXP); END;
           END;
      GOTO CONT;
      END;
DIG:  LDIG=SUBSTR(ACCUM,1,COUNT);
      IF MODE=7 THEN LDIG=-LDIG;
      FSTOR(BNUM)=LDIG; CADD=NCONS(MONE,BNUM);
      BNUM=BNUM-1;
      L=NSTACK(CADD); GOTO CONT;
SPEC: IF MODE=6 THEN DO; PLEV=PLEV+1; GOTO CNT; END;
      IF MODE=4 THEN DO; L=1; RETURN(L); END;
      IF MODE=16 THEN GOTO CONT;
```

```
        IF MODE=7 THEN
             DO; PLEV=PLEV-1;
             IF NEVAL3<0 THEN RETURN(SEXP);
             GOTO CONT;
             END;
        IF MODE=2 THEN DO; DOTTAG='1'B; GOTO CNT; END;
        IF MODE=3 THEN DO; MODETAG='0'B; RETURN(N127); END;
        DISPLAY('ILLEGAL SPECIAL CHARACTER IN S-EXPRESSION');
        RETURN(N127);
  CNT:  L=NSTACK(MODE);
        GOTO CONT;
EVAL1:  ENTRY;

 /* EVALUATES AN ATOM S-EXPRESSION */

        MODETAG=¬MODETAG; L=NUNSTK;
        SEXP=L;
        RETURN;

NEVAL2: ENTRY BIN FIXED;

        L=NSTACK(CADD);
        IF L=200 THEN
             DO; CALL EVAL1;
             RETURN(MONE);
             END;
        RETURN(SEXP);

NEVAL3: ENTRY BIN FIXED;

/* CALLED UPON SCANNING A ')'. RETURNS A "-1" ON PLEV=0. */

        IF DOTTAG THEN GOTO DOTT;
        L1=NUNSTK;
        IF L1=6 THEN    /* NIL LIST: () */
             DO; L1=NIL;
             L=NSTACK(L1); IF PLEV=0 THEN
                  DO;  MODETAG=¬MODETAG;
                       SEXP=L1;
                       RETURN(MONE); END;
             RETURN(L1);
             END;
        IF L1=2 THEN GOTO CH;
        SEXP=NCONS(L1,NIL);
   CH:  DO WHILE('1'B); L1=NUNSTK;
        IF L1=6 THEN
```

```
                DO; IF PLEV=0 THEN
                     DO; MODETAG=¬MODETAG; RETURN(MONE); END;
                L1=NSTACK(SEXP); RETURN(L1);
                END;
          SEXP=NCONS(L1,SEXP);
          END;
 DOTT:    L1=NUNSTK; L2=NUNSTK;
          IF L2¬=2 THEN
                DO; DOTTAG='0'B; GOTO CH;
                END;
          L=NUNSTK;
          SEXP=NCONS(L,L1);
                L1=NUNSTK; IF PLEV=0 THEN
                     DO; MODETAG=¬MODETAG; RETURN(MONE);
                     END;
          L1=NUNSTK; L=NSTACK(L1);
          IF L1=2 THEN DOTTAG='1'B;
               ELSE DOTTAG='0'B;
                L1=NSTACK(SEXP); RETURN(L1);

NSTACK:  ENTRY(J3) BIN FIXED;
          DECLARE
          J3 BIN FIXED,
          L1 BIN FIXED;

          STAC(CTR)=J3;
          L1=CTR;
          CTR=CTR-1;
          RETURN(L1);

NUNSTK:  ENTRY BIN FIXED;
          DECLARE
          J4 BIN FIXED;

          CTR=CTR+1;
          J4=STAC(CTR);
          STAC(CTR)=0;
          RETURN(J4);

/* LEXICAL ANALYSIS PHASE */

 SCAN:   ENTRY;
          DECLARE
          ALPBASE BIN FIXED EXTERNAL,
          CHARSET CHAR(52) EXTERNAL,
          DIGBASE BIN FIXED EXTERNAL,
```

```
          FF BIN FIXED STATIC,
          KK BIN FIXED STATIC,
          LCOUNT BIN FIXED EXTERNAL,
          NO BIN FIXED,
          NUMBERSW(0:3) LABEL (SCANMANT,SCANFRAC,SCANEXSN,SCANEX)
                        INITIAL (SCANMANT,SCANFRAC,SCANEXSN,SCANEX),
          PNAME BIN FIXED EXTERNAL,
          READER ENTRY(BIN FIXED,CHAR(72)) RETURNS(BIT(1)),
          SWITCH(0:3) LABEL (DEBLANK,IDENT,DIGIT,QNUMBER)
                      INITIAL (DEBLANK,IDENT,DIGIT,QNUMBER),
          TT CHAR(1) STATIC:

          RESULT,COUNT,FF,MODE=0;
          ACCUM=' ';
          DO WHILE('1'B);
               BP=BP+1;
               IF BP>72 THEN
               DO; IF ¬READER(LCOUNT,BUFFER) THEN
                    DO; RESULT,CCUNT=0; RETURN;
                    END;
               LCOUNT=LCOUNT+1; BP=1;
               END;
          TT=SUBSTR(BUFFER,BP,1); KK=INDEX(CHARSET,TT);
          GOTO SWITCH(RESULT);
DEBLANK:  IF KK>1 THEN
               IF KK>ALPBASE THEN
                    IF KK>DIGBASE THEN RESULT=2;
                    ELSE RESULT=1;
               ELSE IF KK=7 | KK=8 THEN    /* +-.*/
                    DO; RESULT=3; MODE=KK;
                    END;
                    ELSE    /* SPECIAL CHARACTER*/
                    DO; MODE=KK; COUNT=1; SUBSTR(ACCUM,1,1)=TT;
                    RESULT=3; GOTO RETURNL;
                    END;
          ELSE GOTO XIT;
          GOTO STORET;
IDENT:  IF KK>ALPBASE THEN GOTO STORET;
          GOTO BACKUP;
DIGIT: GOTO NUMBERSW(FF);
 SCANMANT: IF KK>DIGBASE THEN GOTO STORET;
          IF KK=2 THEN    /* DECIMAL POINT */
               DO; MODE=1; FF=1;
               GOTO STORET;
               END;
          ELSE GOTO CHECKEXP;
```

```
 SCANFRAC: IF KK>DIGBASE THEN GOTO STORET;
 CHECKEXP: IF TT='E' THEN
              DO; FF=2; MODE=1;
              GOTO STORET;
              END;
           GOTO BACKUP;
 SCANEXSN: IF KK=7 | KK=8 THEN
           DO; FF=3; GOTO STORET;
           END;
           IF KK>DIGBASE THEN
                DO; FF=3; GOTO STORET;
                END;
           GOTO BACKUP;
  SCANEX: IF KK>DIGBASE THEN GOTO STORET;
           GOTO BACKUP;
 QNUMBER: IF KK>DIGBASE THEN
                DO; RESULT=2;
                IF MODE=5 THEN
                     DO; MODE=1; FF=1;
                     END;
                ELSE MODE=0;
                GOTO STORET;
                END;
           GOTO BACKUP;
STORET: IF COUNT>30 THEN GOTO RETURNL;
           COUNT=COUNT+1;
           SUBSTR(ACCUM,COUNT,1)=TT;
   XIT: END;
BACKUP: BP=BP-1;
RETURNL: RETURN;

 READER: ENTRY(LC,INPT) BIT(1);
           DECLARE
           INPT CHAR(72),
           LC BIN FIXED,
           VAR CHAR(25) VARYING;

           IF TCOUNT=0 THEN VAR='CALL EVALQUOTE, ARGS:';
                ELSE VAR=' ';
           TCOUNT=1; INPT=' ';
           DISPLAY(VAR) REPLY(INPT);
           GO TO GO;
           DISPLAY(LC||'  '||INPT);
     GO: IF INPT='END LISP' THEN RETURN('0'B);
                ELSE RETURN('1'B);
```

```
ENTER: ENTRY(J1);

/* THIS PROCEDURE RECEIVES THE LOCATION IN OBLIST OF AN ATOM AND
   THEN SETS UP THE ATOM WITH ITS PRINT NAME STRING, SETTING
   THE ADDRESS OF ITS HEADER CELL TO THE EXTERNAL VARIABLE 'CADD'. */

      DECLARE
      FSTOR(0:16000) BIN FIXED(31) EXTERNAL,
      J1 BIN FIXED,
      LB32 BIT(32) ALIGNED,
      PN BIN FIXED,
      MCOUNT FIXED BIN,
      TWO16 FIXED BIN (31) STATIC INITIAL(65536),
      TE BIN FIXED;

      FSTOR(J1)=FSTOR(J1)+TWO16;
      L=NIL; MCOUNT=COUNT-1;
      DO I=0 TO MCOUNT;
           K=COUNT-I; TE=UNSPEC(SUBSTR(ACCUM,K,1));
           L=NCONS(TE,L);
      END;
      PN=NCONS(NCONS(MONE,NCONS(PNAME,NCONS(L,NIL))),NIL);
      CADD=NCAR(PN); /* CADD=ADDRESS OF THE ATOM HEADER CELL */
      IF NCAR(J1)=1 THEN L=NRPLACD(J1,PN);
      ELSE DO;
           L=NCDR(J1);
           DO WHILE (NCDR(L)¬=NIL); L=NCDR(L); END;
           K=NRPLACD(L,PN);
           END;
      RETURN;

LOOKUP: ENTRY(J2) BIN FIXED;

/* THIS PROCEDURE RECEIVES A LOCATION IN OBLIST AND THEN LOOKS AT ALL
   THE ATOMS STRUNG FROM THAT LOCATION. IF IT FINDS A MATCH WITH
   THE ATOM STORED IN ACCUM THEN IT RETURNS A "1", SETTING CADD
   EQUAL TO THE ATOM HEADER CELL, ELSE IT RETURNS A "-1".  */

      DECLARE
      II BIN FIXED STATIC,
      J2 BIN FIXED,
      (CP1,CP2) CHAR(1),
      TE1 BIN FIXED;

      NO=NCAR(J2);
      IF NO=0 THEN RETURN(MONE);
```

```
          TE1=NCDR(J2);
          DO II=1 TO NO;
               TE=NCAR(TE1);
               LL=NGET(TE,PNAME);
               KK=1;
               DO WHILE (KK<=COUNT);
                    CP1=SUBSTR(ACCUM,KK,1);
                    UNSPEC(CP2)=SUBSTR(UNSPEC(FSTOR(LL)),9,8);
                    IF CP1¬=CP2 THEN GO TO LOOKA;
                    KK=KK+1; LL=NCDR(LL);
                    IF LL=NIL THEN GO TO LOOKA;
               END;
     LOOKA:    IF KK=COUNT+1 THEN
                    DO; CADD=TE; TE=1; RETURN(TE); END;
               TE1=NCDR(TE1);
          END;
          RETURN(MONE);

END LISPC;
```

```
  /* LISPD - EVALQUOTE,INTERPRETER FUNCTIONS, PRINT AND TRACE
     ROUTINES AND FSUBR FUCTIONS */

NEVALQ: PROC(FNEVQ,AEVQ);

             DECLARE
             FSTOR(0:16000) FIXED BIN(31) STATIC EXT,
             (T,F,NIL,PNAME,APVAL,EXPR,SUBR,FSUBR,FEXPR,BFREE,FLAG,
             LABEL,FNCTION,LAMBDA,FUNARG,COND,QUOTE,DOLLAR,SLASH,LPAR,
             RPAR,COMMA,PERIOD,PLUSS,DASH,STAR,BLANK,EQSIGN)
                  FIXED BIN STATIC EXT,
             (ONE INITIAL(1), TWO INITIAL(2),THREE INITIAL(3),
             FOUR INITIAL(4), FIVE INITIAL(5)) FIXED BIN STATIC,
             (EVAL,EVLIS,EVCON,APPLY) FIXED BIN EXT,
             CEVALQUOTE CHAR(9) INITIAL('EVALQUOTE') STATIC,
             CEVAL CHAR(4) INITIAL('EVAL') STATIC,
             CAPPLY CHAR(5) INITIAL('APPLY') STATIC,
             CEVLIS CHAR(5) INITIAL('EVLIS') STATIC,
             CEVCON CHAR(5) INITIAL('EVCON') STATIC,
             (NBLKS,ERFLAG,BUFFCON,RECUR) FIXED BIN STATIC ,
             PRBUFF CHAR(128) STATIC,
             PRNAME RETURNS(CHAR(30) VARYING),
             (NULL,LENGTH,LAST) ENTRY EXT,
             (TRACE,TRACE1) FIXED BIN EXT INITIAL(0);

             NBLKS=1; RECUR=0; PRBUFF=(128)' ';
             ERFLAG=0;

 /* EVALQUOTE */

             DCL (FNEVQ,AEVQ,VAL1) FIXED BIN;
             IF NATOM(FNEVQ)=T THEN

              /* CHECK FOR FEXPR OR FSUBR */

                  DO;  IF NGET(FNEVQ,FEXPR)¬=NIL THEN
                            DO; VAL1=NEVAL(NCONS(FNEVQ,AEVQ),NIL);
                                GO TO EXEVQ; END;
                       IF NGET(FNEVQ,FSUBR)¬=NIL THEN
                            DO;  VAL1=NEVAL(NCONS(FNEVQ,AEVQ),NIL);
                                 GO TO EXEVQ; END;
                  END;
             VAL1=NAPPLY(FNEVQ,AEVQ,NIL);
  EXEVQ:     IF ERFLAG>0 THEN VAL1=NIL;
             DISPLAY('    ');
             DISPLAY('VALUE IS:');
```

```
            RETURN(VAL1);        /* RETURN TO LISPA (SUPERVISOR) */

/* APPLY */ NAPPLY: PROC(FN,ARGS,NALST) RECURSIVE;
      /* BINDS VARIABLES AND FUNCTION NAMES ON THE A-LIST AND
       HANDLES FUNARG DEVICE (FUNCTIONAL ARGUMENTS) */
        DCL (ABIND,FN,ARGS,NALST,CARFN,NEXPR,NSUBR,VAL2)FIXED BIN,
        (TRCA,TRC1) FIXED BIN INITIAL(0),
        FNCTN CHAR(30) VARYING;
        IF NPROP(APPLY,FLAG,NIL)¬=NIL THEN
             DO; TRCA=1;
                 CALL PTRACE(CAPPLY,FN,ARGS,NALST,THREE,TWO);
             END;
        IF FN=NIL THEN
             DO; VAL2=NIL; GO TO EXAP; END;
        IF NATOM(FN)=T THEN

        /* LOOK FOR FUNCTION DEFINITION ON PROPERTY LIST */

             DO; FNCTN=PRNAME(FN);
/*SUBR*/       NSUBR=NGET(FN,SUBR);
               IF NSUBR¬=NIL THEN
               DO; IF NPROP(FN,FLAG,NIL)¬=NIL THEN TRACE1=T;
                   VAL2=NPROC(NSUBR,ARGS,F,FNCTN);
                   GO TO EXAP; END;
/*EXPR*/       NEXPR=NGET(FN,EXPR);
               IF NEXPR¬=NIL THEN
               DO; IF NPROP(FN,FLAG,NIL)¬=NIL THEN
                   DO; CALL PTRACE(FNCTN,ARGS,NIL,NIL,ONE,TWO);
                       TRC1=1; END;
                    VAL2=NAPPLY(NEXPR,ARGS,NALST);
                    IF TRC1=1 THEN
                    DO;  CALL PTRACE(FNCTN,VAL2,NIL,NIL,
                         ONE,ONE); TRC1=0; END;
                    GO TO EXAP;
               END;

        /* LOOK FOR BINDING OF FUNCTION ON A-LIST */

               ABIND=NSASSOC(FN,NALST,TWO);
               IF ABIND=TWO THEN
               DO;  ERFLAG=2;
               DISPLAY('UNDEFINED FUNCTION - APPLY');
                    GO TO EXAP;
               END;
               VAL2=NAPPLY(NCDR(ABIND),ARGS,NALST);
               GO TO EXAP;
```

```
                    END;
               CARFN=NCAR(FN);

               /* CHECK FOR SPECIAL FORMS */

 /*LAMBDA*/ IF NEQ(CARFN,LAMBDA)=T THEN
               /* BIND VARIABLES AND ARGUMENTS ON A-LIST */
               DO; VAL2=NEVAL(NCADDR(FN),NCONC(NPAIR(NCADR(FN),ARGS),
                    NALST)); GO TO EXAP; END;

/* LABEL */  IF NEQ(CARFN,LABEL)=T THEN
               /* CONS THE FUNCTION NAME AND DEFINITION, ADD TO A-LIST,
                  AND CALL APPLY  */
                 DO; VAL2=NAPPLY(NCADDR(FN),ARGS,NCONS(NCONS(NCADR(FN),
                      NCADDR(FN)),NALST));GO TO EXAP; END;

/*FUNARG*/   IF NEQ(CARFN,FUNARG)=T THEN
                    DO; VAL2=NAPPLY(NCADR(FN),ARGS,NCADDR(FN));
                         GO TO EXAP; END;

               VAL2=NAPPLY(NEVAL(FN,NALST),ARGS,NALST);

     EXAP:     IF ERFLAG>0 THEN VAL2=NIL;
               IF TRCA=1 THEN DO;
                    CALL PTRACE(CAPPLY,VAL2,NIL,NIL,ONE,ONE);
                    TRCA=0;
                    END;
               RETURN(VAL2);
               END NAPPLY;

/* EVAL */   NEVAL: PROC(FORM,ALST) RECURSIVE;
          /* EVALUATES FORMS */
               DCL ( TRCE , NCEV,TREX) FIXED BIN;
               DCL (FORM, ALST, ALBND,NAP,CFORM,ALBD,VAL3,INDIC,TREX)
                    FIXED BIN, FNCTN CHAR(30) VARYING;
               IF NPROP(EVAL,FLAG,NIL)¬=NIL THEN
                    DO; TRCE=1;
                        CALL PTRACE(CEVAL,FORM,ALST,NIL,TWO,TWO);
                        END;
/* NIL */    IF FORM=NIL THEN DO; VAL3=NIL; GO TO EXEV; END;
/*NUMBER*/  IF NUMBERP(FORM)=T THEN DO; VAL3=FORM; GO TO EXEV; END;
/* ATOM */   IF NATOM(FORM)=T THEN

                    /* LOOK FOR VALUE BINDING ON PROP. LIST */

               DO;  NAP=NGET(FORM,APVAL);
```

```
                    IF NAP¬=NIL THEN DO; VAL3=NCAR(NAP); GO TO EXEV; END;

                    /* LOOK FOR VALUE BINDING ON A-LIST */

                    ALBND=NSASSOC(FORM,ALST,THREE);
                    IF ALBND=THREE THEN
                    DO;  ERFLAG=3;
                         DISPLAY('UNBOUND VARIABLE - EVAL');
                         GO TO EXEV;
                    END;
                    VAL3=NCDR(ALBND);  GO TO EXEV;
               END;
               CFORM=NCAR(FORM);

               /* CHECK FOR SPECIAL FORMS */

 /*QUOTE*/     IF CFORM=QUOTE THEN
               DO;  VAL3=NCADR(FORM); GO TO EXEV; END;
/*FUNCTION*/IF CFORM=FUNCTION THEN
               DO; VAL3=LIST3(FUNARG,NCADR(FORM),ALST);
                    GO TO EXEV; END;
 /* COND */  IF CFORM=COND THEN
               DO;  VAL3=NEVCON(NCDR(FORM),ALST); GO TO EXEV; END;

               /* TEST FOR PROG HERE WHEN IT IS IMPLEMENTED */

               IF NATOM(CFORM)=T THEN
               DO; FNCTN=PRNAME(CFORM);

               /* LOOK FOR FUNCTION DEFINITION ON PROP. LIST */

/*SUBR*/            INDIC=NGET(CFORM,SUBR); IF INDIC¬=NIL THEN
                    DO; IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN TRACE1=T:
                        VAL3=NPROC(INDIC,NEVLIS(NCDR(FORM),ALST),
                         F,FNCTN); GO TO EXEV; END;
/*EXPR*/            INDIC=NGET(CFORM,EXPR); IF INDIC¬=NIL THEN
                    DO;  NCEV=NEVLIS(NCDR(FORM),ALST);
                         IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN
                         DO;  TREX=1;
                              CALL PTRACE(FNCTN,NCEV,NIL,NIL,ONE,TWO);
                         END;
                         VAL3=NAPPLY(INDIC,NCEV,ALST);
                         IF TREX=1 THEN
                         DO;  TREX=0;
                              CALL PTRACE(FNCTN,VAL3,NIL,NIL,ONE,ONE);
                         END;
```

```
                    GO TO EXEV;
               END;
/*FSUBR*/      INDIC=NGET(CFORM,FSUBR); IF INDIC¬=NIL THEN
               DO; IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN TRACE1=T;
                    VAL3=NPROC(INDIC,NCDR(FORM),ALST,FNCTN);
                    GO TO EXEV; END;
/*FEXPR*/      INDIC=NGET(CFORM,FEXPR); IF INDIC¬=NIL THEN
               DO; IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN
                    DO; TRFEX=1;
                      CALL PTRACE(FNCTN,NCDR(FORM),NIL,NIL,ONE,TWO);
                    END;
                    VAL3=NAPPLY(INDIC,LIST2(NCDR(FORM),ALST),
                         ALST);
                    IF TRFEX=1 THEN
                    DO; TREX=0;
                      CALL PTRACE(FNCTN,VAL3,NIL,NIL,ONE,ONE);
                    END;
                    GO TO EXEV;
               END;

          /* LOOK FOR FUNCTION DEFINITION ON A-LIST */

                    ALBD=NSASSOC(CFORM,ALST,ONE);
                    IF ALBD=ONE THEN
                    DO;  ERFLAG=1;
          DISPLAY('UNDEFINED FUNCTION - EVAL');
                         GO TO EXEV;
                    END;
                    VAL3=NEVAL(NCONS(NCDR(ALBD),NCDR(FORM)),ALST);
                    GO TO EXEV;
               END;
               VAL3=NAPPLY(NCAR(FORM),NEVLIS(NCDR(FORM),ALST),ALST);
     EXEV:IF ERFLAG>0 THEN VAL3=NIL;
               IF TRCE=1 THEN DO;
                    CALL PTRACE(CEVAL,VAL3,NIL,NIL,ONE,ONE);
                    TRCE=0;
                    END;
               RETURN(VAL3);
               END NEVAL;

/* EVLIS */  NEVLIS: PROC(MEV,AEV) RECURSIVE;
          /* EVALUATES ITEMS IN A LIST */
               DCL (MEV,AEV,VAL4) FIXED BIN;
               IF MEV=NIL THEN RETURN(NIL);
               RETURN(NCONS(NEVAL(NCAR(MEV),AEV),NEVLIS(NCDR(MEV),AEV)));
               END NEVLIS;
```

```
/* EVCON */  NEVCON: PROC(CEVC,AEVC) RECURSIVE;
        /* EVALUATES CONDITIONAL FORMS */
             DCL (CEVC,AEVC,VAL5) FIXED BIN;
             IF CEVC=NIL THEN
             DO;  ERFLAG=4;
             DISPLAY('CONDITIONAL UNSATISFIED - EVCON');
                  GO TO EXEVC;
             END;
             VAL5=NEVAL(NCAAR(CEVC),AEVC);
             IF VAL5¬=F THEN
                  IF VAL5¬=NIL THEN
                  DO;  VAL5=NEVAL(NCADAR(CEVC),AEVC);
                       GO TO EXEVC;
                  END;
             VAL5=NEVCON(NCDR(CEVC),AEVC);
       EXEVC: IF ERFLAG>0 THEN VAL5=NIL;
             RETURN(VAL5);
             END NEVCON;


NPROC: PROC(IX,NARGS,FINDX,FNCTN) RECURSIVE;
        /* RETURNS VALUE OF SYSTEM SUBR AND FSUBR FUNCTIONS.
             NUMBER OF SYSTEM FUNCTIONS IS LIMITED TO 255 WHICH
             IS THE MAX NUMBER REPRESENTABLE IN 8 BITS */
             DCL (IX,NARGS,FINDX,IND,KARGS,ARG(3),VAL6,JX,J)FIXED BIN;
             DCL A(128) LABEL ;
             DCL FNCTN CHAR(30) VARYING, AR(3) LABEL ;
             IF FINDX¬=F THEN      /* FSUBR */
             DO;  IND=SUBSTR(UNSPEC(FSTOR(IX)),9,8);
                  IF TRACE1=T THEN
                       CALL PTRACE(FNCTN,NARGS,NIL,NIL,ONE,TWO);
                  GO TO A(IND);
             END;

        /*   SPREAD ARGUMENTS INTO STANDARD CELLS ARG(1),ARG(2),ETC  */

             KARGS=NARGS;
             J,JX=1;
        SPRD: IF KARGS=NIL THEN GO TO INDEX;
             ARG(J)=NCAR(KARGS);
             KARGS=NCDR(KARGS);       /* MAX NO. ARGS = 3*/
             J=J+1; GO TO SPRD;

             /* GET SUBR NUMBER FROM CAR OF CELL 'IX' */
```

```
INDEX:   IND=SUBSTR(UNSPEC(FSTOR(IX)),9,8);
         IF TRACE1=T THEN DO; JX=J-1; GO TO AR(JX); END;
         ELSE GO TO A(IND);
  AR(1):  CALL PTRACE(FNCTN,ARG(1),NIL,NIL,ONE,TWO);
          GO TO A(IND);
  AR(2):  CALL PTRACE(FNCTN,ARG(1),ARG(2),NIL,TWO,TWO);
          GO TO A(IND);
  AR(3):  CALL PTRACE(FNCTN,ARG(1),ARG(2),ARG(3),THREE,TWO);
          GO TO A(IND);

/* CONS */       A(1):  VAL6=NCONS(ARG(1),ARG(2)); GO TO EXPROC;
/* CAR */        A(3):  VAL6=NCAR(ARG(1)); GO TO EXPROC;
/* CDR */        A(2):  VAL6=NCDR(ARG(1)); GO TO EXPROC;
/* EQ */         A(4):  VAL6=NEQ(ARG(1),ARG(2)); GO TO EXPROC;
/* ATOM */       A(5):  VAL6=NATOM(ARG(1)); GO TO EXPROC;
/* CAAR */       A(6):  VAL6=NCAAR(ARG(1)); GO TO EXPROC;
/* CADR */       A(7):  VAL6=NCADR(ARG(1)); GO TO EXPROC;
/* CDAR */       A(8):  VAL6=NCDAR(ARG(1)); GO TO EXPROC;
/* CDDR */       A(9):  VAL6=NCDDR(ARG(1)); GO TO EXPROC;
/* CAAAR */      A(10): VAL6=NCAAAR(ARG(1)); GO TO EXPROC;
/* CAADR */      A(11): VAL6=NCAADR(ARG(1)); GO TO EXPROC;
/* CADAR */      A(12): VAL6=NCADAR(ARG(1)); GO TO EXPROC;
/* CDAAR */      A(13): VAL6=NCDAAR(ARG(1)); GO TO EXPROC;
/* CADDR */      A(14): VAL6=NCADDR(ARG(1)); GO TO EXPROC;
/* CDADR */      A(15): VAL6=NCDADR(ARG(1)); GO TO EXPROC;
/* CDDAR */      A(16): VAL6=NCDDAR(ARG(1)); GO TO EXPROC;
/* CDDDR */      A(17): VAL6=NCDDDR(ARG(1)); GO TO EXPROC;
/* RPLACA */     A(18): VAL6=NRPLACA(ARG(1),ARG(2)); GO TO EXPROC;
/* RPLACD */     A(19): VAL6=NRPLACD(ARG(1),ARG(2)); GO TO EXPROC;
/* NULL */       A(20): VAL6=NULL(ARG(1)); GO TO EXPROC;
/* EQUAL */      A(21): VAL6=NEQUAL(ARG(1),ARG(2)); GO TO EXPROC;
/* APPEND */     A(22): VAL6=NAPPEND(ARG(1),ARG(2)); GO TO EXPROC;
/* COPY */       A(23): VAL6=NCOPY(ARG(1)); GO TO EXPROC;
/* MEMBER */     A(24): VAL6=MEMBER(ARG(1),ARG(2)); GO TO EXPROC;
/* PAIRLIS */    A(25): VAL6=NPAIRLS(ARG(1),ARG(2),ARG(3));
                        GO TO EXPROC;
/* ASSOC */      A(26): VAL6=NASSOC(ARG(1),ARG(2)); GO TO EXPROC;
/* SUB2 */       A(27): VAL6=NSUB2(ARG(1),ARG(2)); GO TO EXPROC;
/* SUBLIS */     A(28): VAL6=NSUBLIS(ARG(1),ARG(2)); GO TO EXPROC;
/* SASSOC */     A(29): VAL6=NSASSOC(ARG(1),ARG(2),ARG(3));
                        GO TO EXPROC;
/* NCONC */      A(30): VAL6=NCONC(ARG(1),ARG(2)); GO TO EXPROC;
/* ATTRIB */     A(31): VAL6=NATTRIB(ARG(1),ARG(2)); GO TO EXPROC;
/* SUBST */      A(32): VAL6=NSUBST(ARG(1),ARG(2),ARG(3));
                        GO TO EXPROC;
```

```
/* PROP */       A(33): VAL6=NPROP(ARG(1),ARG(2),ARG(3));
                        GO TO EXPROC;
/* GET */        A(34): VAL6=NGET(ARG(1),ARG(2)); GO TO EXPROC;
/* PAIR */       A(35): VAL6=NPAIR(ARG(1),ARG(2));
                        IF VAL6=NIL THEN
                        DO;  ERFLAG=5;
            DISPLAY('ARGUMENTS NOT LISTS OF EQUAL LENGTH - PAIR');
                        END;
                        GO TO EXPROC;
/* NUMBERP */    A(36): VAL6=NUMBERP(ARG(1)); GO TO EXPROC;
/* EFFACE */     A(37): VAL6=NEFFACE(ARG(1),ARG(2)); GO TO EXPROC;
/* LENGTH */     A(38): VAL6=LENGTH(ARG(1)); GO TO EXPROC;
/* LAST */       A(39): VAL6=LAST(ARG(1)); GO TO EXPROC;
/* DEFINE */     A(40): VAL6=NDEFINE(ARG(1)); GO TO EXPROC;
/* DEFLIST */    A(41): VAL6=NDEFLST(ARG(1),ARG(2)); GO TO EXPROC;
/* CSET */       A(42): VAL6=NCSET(ARG(1),ARG(2)); GO TO EXPROC;
/* REMPROP */    A(43): VAL6=NREMPRP(ARG(1),ARG(2)); GO TO EXPROC;
/* FLAG */       A(44): VAL6=NFLAG(ARG(1),ARG(2)); GO TO EXPROC;
/* REMFLAG */    A(45): VAL6=NREMFLG(ARG(1),ARG(2)); GO TO EXPROC;
/* REVERSE */    A(46): VAL6=NREVERS(ARG(1)); GO TO EXPROC;
/* NOT */        A(47): VAL6=NOT(ARG(1)); GO TO EXPROC;
/* PRIN1 */      A(48): VAL6=NPRIN1(ARG(1)); GO TO EXPROC;
/* TERPRI */     A(49): VAL6=NTERPRI; GO TO EXPROC;
/* PRINT */      A(50): VAL6=NPRINT(ARG(1)); GO TO EXPROC;
/* READ */       A(51): VAL6=NREAD(ARG(1)); GO TO EXPROC;
/* EVAL */       A(52): VAL6=NEVAL(ARG(1),NIL); GO TO EXPROC;
/* APPLY */      A(53): VAL6=NAPPLY(ARG(1),ARG(2),NIL);
                        GO TO EXPROC;
/* TRACE */      A(54):  VAL6=NTRACE(ARG(1)); GO TO EXPROC;
/* UNTRACE */    A(55):  VAL6=NUNTRCE(ARG(1)); GO TO EXPROC;
/* ADD1 */       A(56):  VAL6=NADD1(ARG(1)); GO TO EXPROC;
/* SUB1 */       A(57):  VAL6=NSUB1(ARG(1)); GO TO EXPROC;
/* AND */        A(58):  VAL6=NAND(NARGS,FINDX); GO TO EXPROC;
/* OR */         A(59):  VAL6=NOR(NARGS,FINDX); GO TO EXPROC;
/* CSETQ */      A(60):  VAL6=NCSETQ(NARGS,FINDX); GO TO EXPROC;
/* LIST */       A(61):  VAL6=LIST(NARGS,FINDX); GO TO EXPROC;

 EXPROC:  IF TRACE1=T THEN CALL PTRACE(FNCTN,VAL6,NIL,NIL,ONE,ONE);
          TRACE1=F;
          RETURN(VAL6);


 END NPROC;

/* LISP FUNCTIONS WHICH REQUIRE ACCESS TO CURRENT A-LIST */
```

```
/* LIST */  LIST: PROC(LSARG,ALS);
        /* RETURNS A LIST OF ITEMS ON LIST LSARG AFTER THEY ARE
             EVALUATED */
             DCL (LSARG,ALS) FIXED BIN;
             IF NCAR(LSARG)=QUOTE THEN RETURN(NCADR(LSARG));
             RETURN(NEVLIS(LSARG,ALS));
             END LIST;

/* CSET */  NCSET: PROC(SETOB,SETVAL);
        /* PUTS APVAL INDICATOR ON PROPERTY LIST OF VALUE OF SETOB
             WHICH POINTS TO VALUE OF SETVAL */
             DCL (SETOB,SETVAL,CST) FIXED BIN;
             IF NPROP(SETOB,APVAL,NIL)=NIL THEN
                  CST=NRPLACD(LAST(NCDR(SETOB)),NCONS(APVAL,
                       NCONS(NCONS(SETVAL,NIL),NIL)));
             ELSE CST=NRPLACA(NGET(SETOB,APVAL),SETVAL);
             RETURN(SETOB);
             END NCSET;

/* CSETQ */  NCSETQ: PROC(SETQOB,SETQAL);
        /* WORKS LIKE NCSET BUT THE FIRST ARGUMENT IS QUOTED */
             DCL (SETQOB,SETQVAL,SETQVAL1,SETQAL) FIXED BIN;
             SETQVAL1=NEVAL(NCADR(SETQOB),SETQAL);
             RETURN(NCSET(NCAR(SETQOB),SETQVAL1));
             END NCSETQ;

/* AND */  NAND: PROC(ANDLST,AALS);
        /* EVALUATES EACH ITEM IN THE LIST ANDLST UNTIL ONE EVALUATES
             TO F OR NIL.  RETURNS F IN THIS CASE OR T IF THE END OF
             THE LIST IS REACHED */
             DCL (ANDLST,ANDLST1,AND1,AALS) FIXED BIN;
             ANDLST1=ANDLST;
        AND: IF ANDLST1=NIL THEN RETURN(T);
             AND1=NEVAL(NCAR(ANDLST1),AALS);
             IF AND1=F | AND1=NIL THEN RETURN(F);
             ANDLST1=NCDR(ANDLST1); GO TO AND;
             END NAND;

/* OR */  NOR: PROC(ORLST,OALS);
        /* EVALUATES EACH ITEM IN THE LIST ORLST UNTIL ONE EVALUATES
             TO SOMETHING OTHER THAN T.  RETURNS T IN THIS CASE OR F
             IF ALL ITEMS EVALUATE TO F OR NIL */
             DCL (ORLST,ORLST1,EVALOR,OALS) FIXED BIN;
             ORLST1=ORLST;
        OR:  IF ORLST1=NIL THEN RETURN(F);
             EVALOR=NEVAL(NCAR(ORLST1),OALS);
```

```
            IF EVALOR¬=F THEN
                 IF EVALOR¬=NIL THEN RETURN(T);
            ORLST1=NCDR(ORLST1); GO TO OR;
            END NOR;

 /* OUTPUT FUNCTIONS */

 /* PRINT */ NPRINT: ENTRY(XPRIN);
       /* PRINTS THE S-EXPRESSION PASSED AS THE ARGUMENT */
            DCL (XPRIN,DP1) FIXED BIN;
            PRBUFF=(128)' ';
            BUFFCON=NBLKS;
            DP1=NPRIN(XPRIN);
            DP1=NTERPRI;
            RETURN(XPRIN);

NPRIN: PROC(KPRIN) RECURSIVE;
       /* PUTS REPRESENTATION OF S-EXPRESSION KPRIN IN PRINT BUFFER */
            DCL (DUM1,DUM2,DUM3,JPRIN,KPRIN) FIXED BIN;
            IF NATOM(KPRIN)=T THEN GO TO DP;
            JPRIN=KPRIN;
            SUBSTR(PRBUFF,BUFFCON,1)='('; BUFFCON=BUFFCON+1;
       AP:  DUM2=NPRIN(NCAR(JPRIN));
            IF NCDR(JPRIN)=NIL THEN GO TO CP;
            SUBSTR(PRBUFF,BUFFCON,1)=' '; BUFFCON=BUFFCON+1;
            IF NATOM(NCDR(JPRIN))=T THEN GO TO BP;
            JPRIN=NCDR(JPRIN);
            GO TO AP;
       BP:  SUBSTR(PRBUFF,BUFFCON,1)='.'; BUFFCON=BUFFCON+1;
            SUBSTR(PRBUFF,BUFFCON,1)=' '; BUFFCON=BUFFCON+1;
            DUM3=NPRIN1(NCDR(JPRIN));
       CP:  SUBSTR(PRBUFF,BUFFCON,1)=')'; BUFFCON=BUFFCON+1;
            RETURN(KPRIN);
       DP:  DUM3=NPRIN1(KPRIN);
            RETURN(KPRIN);
            END NPRIN;

 /* PRIN1 */ NPRIN1: PROC(NPRN);
       /* PUTS PRINT NAME OF ATOM 'NPRN' ON PRINT BUFFER */
            DECLARE
            TBUFF CHAR(14) STATIC,
            TBUFF1 CHAR(14) VARYING,
            TPBUFF CHAR(30) STATIC,
            LENGTH BUILTIN,
            CH CHAR(1) STATIC,
            (KPRN,NPRN,DPRN,PC,LC,LPR,LPR1) FIXED BIN ;
```

```
          IF NUMBERP(NPRN)=T THEN   /* IS ATOM A NUMBER? */
          DO;  TBUFF=(14)' ';
               TBUFF=FSTOR(NCDR(NPRN));
               PC=1;
               DO WHILE (SUBSTR(TBUFF,PC,1)=' '); PC=PC+1; END;
               TBUFF1=SUBSTR(TBUFF,PC,15-PC);
               LC=LENGTH(TBUFF1);
               SUBSTR(PRBUFF,BUFFCON,LC)=TBUFF1;
               BUFFCON=BUFFCON+LC;
               RETURN(NPRN);
          END;
          LPR=1; TPBUFF=(30)' ';
          KPRN=NGET(NPRN,APVAL);   /* CHARACTER OBJECT? */
          IF NCAR(KPRN)<128 THEN GO TO NPRA;
          KPRN=NGET(NPRN,PNAME);
     NPRA:DO WHILE (KPRN¬=NIL);
               UNSPEC(CH)=SUBSTR(UNSPEC(FSTOR(KPRN)),9,8);
               SUBSTR(TPBUFF,LPR,1)=CH;
               KPRN=NCDR(KPRN);
               LPR=LPR+1;
          END;
          LPR1=LPR-1;
          IF (LPR1+BUFFCON)>120 THEN DPRN=NTERPRI;
          SUBSTR(PRBUFF,BUFFCON,LPR1)=SUBSTR(TPBUFF,1,LPR1);
          BUFFCON=BUFFCON+LPR1;
          RETURN(NPRN);
          END NPRIN1;

 /* TERPRI */  NTERPRI: PROC;
        /* PRINTS CURRENT CONTENTS OF PRINT BUFFER */
          DISPLAY(PRBUFF);
          PRBUFF=(128)' ';
          BUFFCON=NBLKS;
          RETURN(NIL);
          END NTERPRI;

LIST2: PROC(LARG2,LARG3);
        /* RETURNS A LIST OF THE TWO ARGUMENTS */
          DCL (LARG2,LARG3);
          RETURN(NCONS(LARG2,NCONS(LARG3,NIL)));
          END LIST2;

LIST3: PROC (LARG4,LARG5,LARG6);
        /* RETURNS A LIST OF THE THREE ARGUMENTS */
          DCL (LARG4,LARG5,LARG6);
          RETURN(NCONS(LARG4,NCONS(LARG5,NCONS(LARG6,NIL))));
```

```
          END LIST3;

PRNAME: PROC(JPN) CHAR(30) VARYING;
        /* RETURNS CHARACTER STRING OF PRINT NAME OF FUNCTION 'JPN'
           WHICH IS LIMITED TO 30 CHARACTERS */
           DCL (JPN,PRN1,KPN,I) FIXED BIN,
           FNCTN CHAR(30) VARYING,
           CH CHAR(1);
           PRN1=NGET(JPN,PNAME);
           FNCTN=(30)' ';
           I=1; KPN=PRN1;
           DO WHILE (KPN¬=NIL);
                UNSPEC(CH)=SUBSTR(UNSPEC(FSTOR(KPN)),9,8);
                SUBSTR(FNCTN,I,1)=CH;
                I=I+1; KPN=NCDR(KPN);
           END;
           FNCTN=SUBSTR(FNCTN,1,I-1);
           RETURN(FNCTN);
           END PRNAME;

PTRACE: PROC(FNAME,ARG1,ARG2,ARG3,NGS,IO);
        /* DISPLAYS TRACE INFO ON TERMINAL WHEN FUNCTIONS ARE
           ENTERED AND EXITED - WILL NOW WORK ONLY FOR FUNCTIONS WITH
           LESS THAN FOUR ARGUMENTS  */
          DCL (ARG1,ARG2,ARG3,NGS,IO,LIM,LF) FIXED BIN,
           LENGTH BUILTIN,
           FNAME CHAR(30) VARYING;
           LF=LENGTH(FNAME);
           IF IO=1 THEN GO TO LEAVE;
           IF NBLKS>80 THEN DO; RECUR=RECUR+1; NBLKS=1; END;
           SUBSTR(PRBUFF,NBLKS,12+LF)='CALL '||FNAME||', ARGS:';
           LIM=NTERPRI;
           LIM=NPRINT(ARG1);
           IF NGS=1 THEN GO TO EXPTR;
           LIM=NPRINT(ARG2);
           IF NGS=2 THEN GO TO EXPTR;
           LIM=NPRINT(ARG3);
     EXPTR: NBLKS=NBLKS+4;
           RETURN;
      LEAVE: NBLKS=NBLKS-4;
           IF NBLKS<1 THEN DO;
                IF RECUR>0 THEN DO;
                     RECUR=RECUR-1; NBLKS=80; END;
                ELSE NBLKS=1;
                END;
           SUBSTR(PRBUFF,NBLKS,13+LF)='VALUE OF '||FNAME||' IS:';
```

```
            LIM=NTERPRI;
            LIM=NPRINT(ARG1);
            RETURN;
        END PTRACE;

    END NEVALQ;
```

```
/* LISPE: INITIALIZATION ROUTINE */
INITIAL: PROC;

      DECLARE
      ACCUM CHAR(30) EXTERNAL,
      ALPBASE BIN FIXED EXTERNAL,
      APVAL BIN FIXED EXTERNAL,
      BLANK BIN FIXED EXTERNAL,
      BP BIN FIXED EXTERNAL,
      BNUM FIXED BIN EXT,
      BUFFER CHAR(72) EXTERNAL,
      BUFFNO BIN FIXED EXTERNAL,
      CADD BIN FIXED EXTERNAL,
      CHARSET CHAR(52) EXTERNAL,
      COMMA BIN FIXED EXTERNAL,
      COND BIN FIXED EXTERNAL,
      COUNT BIN FIXED EXTERNAL,
      CTR BIN FIXED EXTERNAL,
      DASH BIN FIXED EXTERNAL,
      DIGBASE BIN FIXED EXTERNAL,
      DOLLAR BIN FIXED EXTERNAL,
      EXPR BIN FIXED EXTERNAL,
      EQSIGN BIN FIXED EXTERNAL,
      (EVAL,APPLY) FIXED BIN EXT,
      F BIN FIXED EXTERNAL,
      FEXPR BIN FIXED EXTERNAL,
      FNCTION BIN FIXED EXTERNAL,
      FREE BIN FIXED EXTERNAL,
      FSUBR BIN FIXED EXTERNAL,
      FLAG BIN FIXED EXTERNAL,
      FSTOR(0:16000) BIN FIXED(31) EXTERNAL,
      FUNARG BIN FIXED EXTERNAL,
      JJ BIN FIXED EXTERNAL,
      JK FIXED BIN,
      LABEL BIN FIXED EXTERNAL,
      LAMBDA BIN FIXED EXTERNAL,
      LCOUNT BIN FIXED EXTERNAL,
      LINE BIN FIXED EXTERNAL,
      LPAR BIN FIXED EXTERNAL,
      MFSTOR FIXED BIN EXT,
      MODETAG BIT(1) EXTERNAL,
      NIL BIN FIXED EXTERNAL,
      NREAD ENTRY EXTERNAL,
      PERIOD BIN FIXED EXTERNAL,
      PLUSS BIN FIXED EXTERNAL,
```

```
  PNAME BIN FIXED EXTERNAL,
  QUOTE BIN FIXED EXTERNAL,
  RPAR BIN FIXED EXTERNAL,
  SLASH BIN FIXED EXTERNAL,
  STAC(200) BIN FIXED EXTERNAL,
  STAR BIN FIXED EXTERNAL,
  STKTAB BIT(1) EXTERNAL,
  SUBR BIN FIXED EXTERNAL,
  SYSPRINT FILE OUTPUT PRINT ENVIRONMENT
       (CONSECUTIVE F(3325,133)),
  T BIN FIXED EXTERNAL,
  TCOUNT FIXED BIN EXTERNAL,
  VAR CHAR(10) VARYING,
  VAR1 CHAR(1),
  VAR2 BIN FIXED EXT;


  DISPLAY('      ');
  DISPLAY('NPS LISP INITIALIZING');
  DISPLAY('      ');
  MFSTOR=16000;
  DO I=1 TO 127; FSTOR(I)=0; END;  /* ZERO OUT OBLIST */
  DO I=128 TO MFSTOR-1; FSTOR(I)=I+1; END;  /* LINK UP */
  FREE=128;
  FSTOR(MFSTOR)=0;
  DO I=1 TO 200; STAC(I)=0; END;
  BNUM=MFSTOR;
  APPLY=1030; EVAL=1018;
CHARSET=' .$&*()-+="'':;?,ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
  ALPBASE=INDEX(CHARSET,'A')-1;
  DIGBASE=INDEX(CHARSET,'0')-1;
  LCOUNT,LINE=1;
  MODETAG,STKTAB='0'B;
  TCOUNT,BUFFNO=0;
  NIL=142; PNAME=135; APVAL=154;      /* BOOTSTRAP VALUES */
  BUFFER='PNAME NIL APVAL EXPR SUBR FEXPR FSUBR COND QUOTE &';
  JK=NREAD;
  BUFFER='SLASH PLUSS DASH STAR BLANK EQSIGN COMMA PERIOD &';
  JK=NREAD;
  BUFFER='DOLLAR RPAR LPAR &';
  JK=NREAD;
  BUFFER='FUNCTION FUNARG LAMBDA LABEL F T FLAG &';
  JK=NREAD;  LINE=1;
 BUFFER='CONS CDR CAR EQ ATOM CAAR CADR CDAR CDDR CAAAR CAADR &';
  JK=NREAD;
  BUFFER='CADAR CDAAR CADDR CDADR CDDAR CDDDR RPLACA RPLACD &';
```

```
        JK=NREAD;
        BUFFER='NULL EQUAL APPEND COPY MEMBER PAIRLIS ASSOC SUB2 &';
        JK=NREAD;
        BUFFER='SUBLIS SASSOC NCONC ATTRIB SUBST PROP GET PAIR &';
        JK=NREAD;
        BUFFER='NUMBERP EFFACE LENGTH LAST DEFINE DEFLIST CSET &';
        JK=NREAD;
        BUFFER='REMPROP FLAG REMFLAG REVERSE NOT &';
        JK=NREAD;
        BUFFER='PRIN1 TERPRI PRINT READ EVAL APPLY &';
        JK=NREAD;
        BUFFER='TRACE UNTRACE ADD1 SUB1 &';
        JK=NREAD;
        BUFFER='AND OR CSETQ LIST &'; JK=NREAD;
        STKTAB='1'B;
      RETURN;

INIT1: ENTRY(I1);

        DECLARE
        SWT(10:20) LABEL(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11)
             INITIAL(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11);

        GOTO SWT(I1);
        A1: VAR1='/'; GOTO A12;    A2: VAR1='+'; GOTO A12;
        A3: VAR1='-'; GOTO A12;    A4: VAR1='*'; GOTO A12;
        A5: VAR1=' '; GOTO A12;    A6: VAR1='='; GOTO A12;
        A7: VAR1=','; GOTO A12;    A8: VAR1='.'; GOTO A12;
        A9: VAR1='$'; GOTO A12;    A10: VAR1=')'; GOTO A12;
        A11: VAR1='(';
A12: L=UNSPEC(VAR1); VAR2=NCONS(L,NIL); JJ=-1; RETURN;

 INITL: ENTRY(I2);     /* SET UP APVALS AND PROGRAM VARIABLE VALUES */
        DECLARE LB(27) LABEL;

        GO TO LB(I2);

        LB(1):     PNAME=CADD; RETURN;
        LB(2):     NIL=CADD; VAR2=NCONS(NIL,NIL); CALL SETAP(NIL);
                     RETURN;
        LB(3):     APVAL=CADD; RETURN;
        LB(4):     EXPR=CADD; RETURN;
        LB(5):     SUBR=CADD; RETURN;
        LB(6):     FEXPR=CADD; RETURN;
        LB(7):     FSUBR=CADD; RETURN;
        LB(8):     COND=CADD; RETURN;
```

```
        LB(9):    QUOTE=CADD; RETURN;
        LB(10):   SLASH=CADD; CALL SETAP(SLASH); RETURN;
        LB(11):   PLUSS=CADD; CALL SETAP(PLUSS); RETURN;
        LB(12):   DASH=CADD; CALL SETAP(DASH); RETURN;
        LB(13):   STAR=CADD; CALL SETAP(STAR); RETURN;
        LB(14):   BLANK=CADD; CALL SETAP(BLANK); RETURN;
        LB(15):   EQSIGN=CADD; CALL SETAP(EQSIGN); RETURN;
        LB(16):   COMMA=CADD; CALL SETAP(COMMA); RETURN;
        LB(17):   PERIOD=CADD; CALL SETAP(PERIOD); RETURN;
        LB(18):   DOLLAR=CADD; CALL SETAP(DOLLAR); RETURN;
        LB(19):   RPAR=CADD; CALL SETAP(RPAR); RETURN;
        LB(20):   LPAR=CADD; CALL SETAP(LPAR); RETURN;
        LB(21):   FNCTION=CADD; RETURN;
        LB(22):   FUNARG=CADD; RETURN;
        LB(23):   LAMBDA=CADD; RETURN;
        LB(24):   LABEL=CADD; RETURN;
        LB(25):   F=CADD; M=NRPLACD(NCDDR(F),NCONS(APVAL,
                       NCONS(NCONS(NIL,NIL),NIL)));
                       RETURN;
        LB(26):   T=CADD; VAR2=NCONS(T,NIL); CALL SETAP(T); RETURN;
        LB(27):   FLAG=CADD; RETURN;
        RETURN;

  SETAP: ENTRY(SAPVL);
        DCL SAPVL FIXED BIN;
        M=NRPLACD(NCDDR(SAPVL),NCONS(APVAL,NCONS(VAR2,NIL)));
        RETURN;
 INITL2: ENTRY(I3);     /* SET UP SUBR INDICATORS */

        LM=NCDDR(CADD);
        M=NRPLACD(LM,NCONS(SUBR,NCONS(NCONS(I3,NIL),NIL)));
        RETURN;

 INITL3: ENTRY(I4);     /* SET UP FSUBR INDICATORS */

        LM=NCDDR(CADD);
        M=NRPLACD(LM,NCONS(FSUBR,NCONS(NCONS(I4,NIL),NIL)));
        RETURN;

END INITIAL;
```

LIST OF REFERENCES

1. Berkeley, E. C., and others, The Programming Language LISP: Its Operation and Applications, Information International, Inc., 1964.

2. McCarthy, J., and others, LISP 1.5 Programmer's Manual, The M.I.T. Press, 1962.

3. System Development Corporation Report TM-2337/101/00, LISP 1.5 Reference Manual for Q-32, by S. L. Kameny, 9 August 1965.

4. Bolt Beranek and Newman, Inc., The BBN 940 LISP System, by D. G. Bobrow, and others, 15 July 1967.

5. Bolt Beranek and Newman, Inc., The Structure of a LISP System Using Two-Level Storage, by D. G. Bobrow and D. L. Murphy, 4 November 1966.

6. Weissman, C., LISP 1.5 Primer, Dickerson Publishing Company, 1967.

7. IBM System/360 PL/I Reference Manual, Form C28-8201-1, International Business Machines, Corp., March 1968.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chief of Naval Operations (OP-91) Department of the Navy Washington, D. C. 20350	1
4.	LTJG Gary A. Kildall, USNR, Code 53Kd (thesis advisor) Department of Mathematics Naval Postgraduate School Monterey, California 93940	2
5.	Asst. Professor G. E. Heidorn (Code 55Hd) Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
6.	LT Donald G. Gentry, USN (student) USS Kitty Hawk (CVA-63) FPO San Francisco 96601	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

3. REPORT TITLE

AN IMPLEMENTATION OF LISP 1.5 FOR THE IBM 360/67 COMPUTER

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; December 1969

5. AUTHOR(S) (First name, middle initial, last name)

Donald G. Gentry

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1969	100	7

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

The design and implementation of the NPS LISP programming system is described. NPS LISP is an interactive version of LISP 1.5, a sophisticated list processing and symbol manipulation computer language. NPS LISP was implemented in PL/I for operation under the CP/CMS time-sharing system on the IBM 360/67 computer. It is an interpretive system patterned after 7090 LISP. Most of the features of 7090 LISP are included in NPS LISP.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
LISP						
List Processing						
List Structure						
LISP Interpreter						
List Processing Computer Programming Language						
Symbol Manipulation Computer Programming Language						
LISP 1.5						